Treball de Fi de Grau

Grau d'enginyeria en tecnologies industrials

Disseny i programació d'un sistema de reconeixement de perfils d'identitat digital

MEMÒRIA

Autor: Sara Ribes Amorós
Director: Álvaro Gómez Pau
Codirector: Salvador Manich Bou

Convocatòria: 01/2019

Escola Tècnica Superior
d'Enginyeria Industrial de Barcelona

Índex

Resum

El present treball de fi de grau s'ha dut a terme al grup de recerca QinE del departament d'Enginyeria Electrònica de l'Escola Tècnica Superior d'Enginyeria Industrial de Barcelona durant el quadrimestre de tardor del curs acadèmic 2018-2019. Aquest treball té com a objectiu reconèixer identitats a partir de patrons de pressió realitzats amb els dits de la mà. Es persegueix el desenvolupament d'un software capaç de reconèixer perfils d'identitat digital de forma dinàmica, ràpida, econòmica i veraç. Aquest objectiu s'assoleix mitjançant el software descrit, juntament amb el hardware desenvolupat pel David Gil en el seu treball de de fi de grau: "Sistema personal de captació i generació de signatura difosa anti-pirateig". Per al desenvolupament del software s'ha fet una recerca d'algorismes d'aprenentatge automàtic o *Machine Learning*, més concretament, dels algorismes d'aprenentatge supervisat posant-hi èmfasi a les xarxes neuronals amb les que s'ha aconseguit resoldre el problema plantejat exitosament. S'ha triat el llenguatge de programació Python degut a la seva gran versatilitat. En el decurs de la memòria es presenten els aspectes més rellevants de la implementació de les xarxes neuronals emprant, així com. els resultats de la xarxa neuronal implementada pel reconeixement d'identitats.

1. Introducció

1.1. Motivació del projecte

En la societat actual, el consum massiu de dispositius electrònics permet als usuaris emmagatzemar tot allò que desitgin: correus electrònics, comptes bancaris, fotografies, documents, etc. La informació que aquests dispositius són capaços d'emmagatzemar creix de tal forma que en un futur no molt llunyà solament un dispositiu electrònic permetrà emmagatzemar tota la informació personal que l'usuari disposi. Degut a la possibilitat de guardar tanta informació important per l'usuari, neix la necessitat d'instaurar una signatura que impedeixi l'accés d'altres usuaris.

Hi ha molts tipus de mètodes moderns per a fer segura la informació, si s'esmenten els més populars de forma cronològica es poden trobar els següents: els més antics són identificacions físiques, emprant targetes o claus, més endavant apareixen els numèrics i/o alfabètics teclejats en un teclat. Davant l'evolució de la tecnologia i la implementació de pantalles tàctils en l'electrònica, es desenvolupen els sistemes d'identificació mitjançant patrons. Finalment, en els últims anys, s'han implementat sistemes de reconeixement més sofisticats: lectura de l'empremta del dit, facial, de retina, d'iris, escriptura, batec del cor, veu, genètic, etc. Aquests, utilitzen sensors biomètrics, els quals realitzen lectures d'atributs o comportaments físics, es poden diferenciar entre ells segons si la lectura es fa sobre un element físic (p. e. lectura de l'empremta), si s'està reconeixent un comportament (p. e. escriptura), o bé si es fa una lectura de quelcom innat o químic (p. e. reconeixement de l'ADN) [1]. En la figura 1.1. es pot veure un esquema dels mètodes d'identificació personal mencionats ordenats cronològicament.

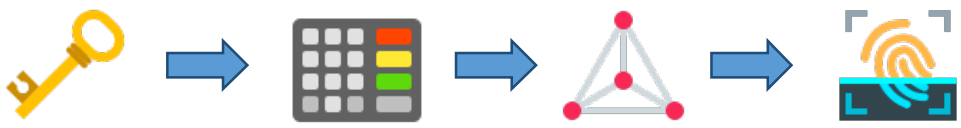


Figura 1.1. Diagrama de l'evolució dels mètodes moderns d'identificació de més antic a més actual. (Font: Pròpia)

Com es pot comprovar, el canvi en el temps de les signatures sempre ha perseguit el mateix objectiu: garantir la seguretat de les dades de l'usuari i generar una sistema d'identificació únic i intransferible. Aquest objectiu ve motivat per garantir la protecció de les dades davant de la pèrdua, robatori o atacs hardware/software mal intencionats.

Enfront el ventall de sistemes d'identificació personal que ofereix el mercat, s'hi detecten una sèrie de febleses. Per una banda, sempre que es tracti d'un objecte físic (claus i targetes) es corre el risc de pèrdua o robatori. Les contrasenyes teclejades es poden reproduir utilitzant càmeres fotogràfiques o tèrmiques. D'altra banda, el reconeixement d'identitats mitjançant sensors biomètrics dóna lloc a identificacions úniques i intransferibles, doncs no poden ser reproduïdes per una altra persona que no sigui l'usuari a identificar. Tot i això, presenten un altre tipus de problema i són els canvis que experimenta el cos humà al llarg de la vida: pèrdua de les empremtes, canvi de l'obertura de l'iris depenent de la intensitat de la llum, canvi de la veu, entre d'altres. Per tant, als objectius dels sistemes de reconeixement de l'usuari, s'afegeix la propietat de ser sistemes dinàmics. Per últim, els sensors biomètrics mesuren característiques innates o químiques i requereixen de processos difícils i costosos fet que dificulta la identificació ràpida i econòmica.

Figura 1.2. Mancances de les contrasenyes que ofereix el mercat actual. (Font: Pròpia)

Degut a les mancances esmentades i representades en la figura 1.2., neix la idea del reconeixement d'un usuari mitjançant un patró de pressió que l'usuari realitza amb els dits de la mà sobre un conjunt de sensors seguint una seqüència que pot ser, per exemple, una melodia, sense aixecar els dits dels sensors fins acabar la seqüència. Amb aquest mètode, s'implementa una contrasenya única, intransferible, duradora, ràpida i econòmica. Un cop trobada la forma de registrar la seqüència generada per l'usuari, el sistema requereix d'un software capaç de processar les dades i reconèixer l'usuari davant la infinitat de contrasenyes possibles.

En resum, la motivació d'aquest treball és desenvolupar un sistema dinàmic, capaç de reconèixer identitats que permeti la validació d'un usuari mitjançant un patró de pressions únic, intransferible, durador, ràpid i econòmic. En la figura 1.3. es recullen les propietats més importants que han de caracteritzar al software.

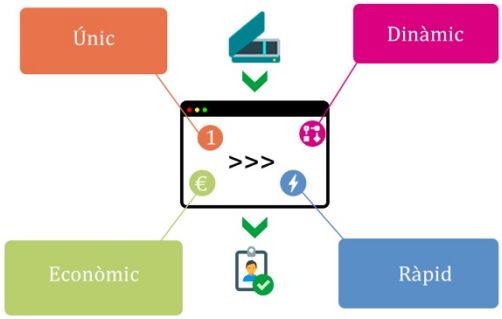


Figura 1.3. Característiques més significatives del software. (Font: Pròpia)

1.2. Objectius del projecte

Davant la problemàtica presentada, es defineixen els següents objectius principals:

1. Desenvolupar un software fiable capaç de reconèixer patrons temporals de pressió i reconèixer els usuaris que els han generat, permetent així el reconeixement d'identitats.

2. Estudiar en profunditat els algorismes que permeten la implementació d'un software amb les característiques esmentades anteriorment.

1.3. Abast del projecte

Aquest treball de fi de grau forma part d'un projecte conjunt amb el David Gil que ha realitzat el següent treball de fi de grau: "Sistema personal de captació i generació de signatura difosa anti-pirateig", aquest es basa en el desenvolupament del hardware per tal de poder fer la verificació d'usuaris a través de patrons de pressió. Per tant, en el projecte es poden distingir

dues parts: la primera, la lectura dels patrons de pressió a través del hardware, en el treball del David Gil i la segona, la verificació de l'usuari a partir d'un software.

Aquesta segona part és en la que es basa aquest projecte, aquest comença a la recepció de les dades, passa pel software encarregat de realitzar la verificació a partir de les dades experimentals i finalitza en la confirmació a l'usuari de l'autenticitat, o no, del patró introduït.

En cap moment s'entrarà en detall sobre el funcionament ni les parts del hardware, simplement per tal d'entendre aquest projecte cal saber que les dades s'obtenen a partir de sensors, aquestes arriben via port USB i també cal conèixer la forma de les dades amb les quals es treballa per obtenir la solució plantejada. En la figura 1.4. es poden veure les parts del projecte de forma esquemàtica.

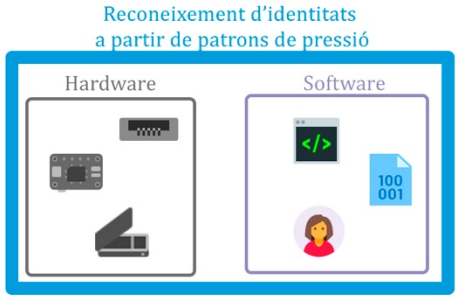


Figura 1.4. Desglossament del projecte. (Font: Pròpia amb icones extretes del recurs https://icons8.com/icons)

2. Estat de l'art

2.1. Introducció

La resolució d'un problema mitjançant un software requereix d'uun algorisme queno és més que una seqüència d'ordres amb les quals a partir d'un input es pot obtenir un output en un temps d'execució finit. Per exemple, en un full de càlcul electrònic, per tal d'ordenar un conjunt de dades alfabèticament, cal seleccionar-les i elegir el botó corresponent per tal de realitzar l'ordre. Darrere d'aquest botó hi ha un algorisme (*quick sort, bubble sort, insertion sort,* etc.) que rep per entrada el conjunt de dades seleccionades i segueix una seqüència d'instruccions de forma que la sortida sigui la mateixa llista de dades ordenades alfabèticament. En la figura 1.5. es pot veure un diagrama de la implementació de l'algorisme descrit en l'exemple.

Figura 2.1. Diagrama d'implementació d'un algorisme específic. (Font: Pròpia)

Hi ha casos, però, en què no es disposa d'un algorisme específic, com és el cas de classificar una contrasenya, obtinguda mitjançant patró de pressions caracteritzat per un valor numèric, segons si aquesta pertany o no a l'usuari a identificar. En el cas anterior es coneix l'input, el valor que simbolitza la contrasenya, també sabem que la resposta pot ser sí, pertany a l'usuari o no, no és l'usuari a identificar, malauradament no es disposa de cap botó que permeti el pas de l'entrada a la sortida, a diferència d'ordenar un conjunt de dades en un full de càlcul. És per a aquest tipus de problemes, més comuns del que es pensa, que la intel·ligència artificial ha trobat el seu lloc en la societat actual.

El que es desitja és que l'ordinador llegeixi un conjunt de contrasenyes que identifiquen a l'usuari i un altre conjunt que pertanyen a altres usuaris i que l'aparell n'extregui un algorisme per tal de validar quines contrasenyes pertanyen a l'usuari i quines no.

Els avenços fets en la tecnologia permeten, a partir d'un dispositiu electrònic, com un ordenador, processar i emmagatzemar un bon munt de dades. Aquesta capacitat és molt útil quan les dades emmagatzemades són processades i convertides en informació que es pugui fer servei, per exemple, convertides en una predicció.

Per exemple, si s'emmagatzema el conjunt de contrasenyes fetes per un mateix usuari, es poden analitzar i intentar buscar patrons repetitius en aquestes. Segurament, davant d'una nova contrasenya es pot identificar algun dels patrons i fent una aproximació, ja que és provable que la nova contrasenya no sigui idèntica a alguna de les emmagatzemades, es pugui afirmar que es tracta de l'usuari a reconèixer. En efecte, aquesta és la base de l'aprenentatge automàtic i els mètodes que utilitzen per tal de trobar un output a partir d'un input s'anomenen algorismes d'aprenentatge estadístic o automàtic. La forma d'implementació d'un algorisme d'aprenentatge està representada en la figura 1.6. mitjançant un diagrama.

Figura 2.2. Diagrama d'implementació d'un algorisme d'aprenentatge automàtic. (Font: Pròpia)

Com es pot veure en la figura 2.3., els algorismes d'aprenentatge formen part de l'aprenentatge automàtic. Aquest, és una branca de la intel·ligència artificial que pertany a l'àrea de la computació i la seva funció és analitzar i interpretar patrons i estructures de les dades a través d'algorismes per tal de poder aprendre, raonar i prendre decisions amb la mínima intervenció humana i de forma satisfactòria (amb el mínim error).

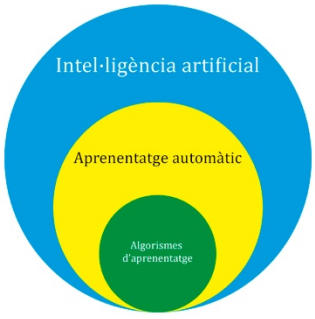


Figura 2.3. Intel·ligència artificial, aprenentatge automàtic i algorismes d'aprenentatge. (Font: pròpia)

De forma general, per tal de dur a terme la seva funció, els algorismes d'aprenentatge necessiten d'un conjunt de dades d'entrenament en forma de vector d'entrada, aquestes poden ser obtingudes de forma experimental o bé ser extretes de bases de dades, seguidament, mitjançant algoritmes computacionals, es prediu un vector de sortida.

2.2. Tipus d'aprenentatge automàtic

Per tal d'elegir l'algorisme d'aprenentatge que més s'adapti al problema en que es basa aquest projecte, s'ha fet una recerca dels diferents sistemes. Com es pot veure en l'esquema de la figura 2.4. els algorismes d'aprenentatge es poden classificar segons les dades obtingudes i la finalitat amb que són utilitzats, poden ser algorismes supervisats, no supervisats, semi supervisats i reforçats [2].

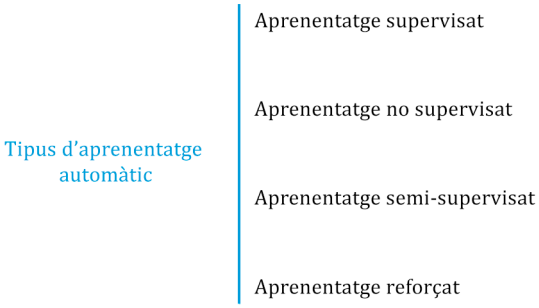


Figura 2.4. Esquema de classificació de l'aprenentatge automàtic. (Font: pròpia)

2.2.1. Aprenentatge supervisat

En aquest cas es parteix de dades d'entrenament categoritzades, és a dir, són exemples en forma de vectors d'entrada, prèviament mesurats o establerts, amb un vector de sortida que es desitja obtenir establert per la persona que prepara les dades d'entrenament. D'aquí el nom de supervisat, aquesta persona actua com a "professor" i "ensenya" les sortides de les dades d'entrada. L'objectiu d'aquest aprenentatge és utilitzar els inputs per tal de predir els outputs, tot comparant-los amb les sortides desitjades. Els outputs, poden ser quantitatius o bé qualitatius.

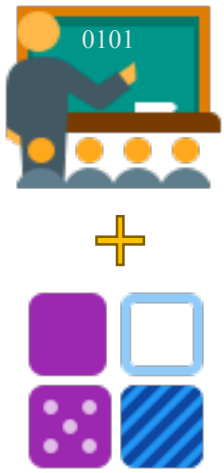
Per exemple, la lectura de dígits manuscrits mitjançant una màquina és un cas d'aprenentatge supervisat, les dades d'entrenament consisteixen en un conjunt de dígits escanejats, escrits per persones aleatòries i posteriorment categoritzats, des del zero fins al nou [3]. Un cop la màquina és entrenada amb aquestes dades, serà capaç de reconèixer un dígit escrit que aquesta no hagi processat anteriorment.

2.2.2. Aprenentatge no supervisat

Molt semblant a l'aprenentatge anterior però en aquest cas les dades d'entrenament no estan categoritzades. En aquest tipus no existeix la figura de "professor" ja que l'objectiu, en aquest cas, és descobrir grups entre els inputs.

Per exemple, distingir diferents races de gossos a partir un conjunt de dades que consisteixin en característiques dels gossos (alçada, llargada, pes, etc.).

2.2.3. Aprenentatge semi supervisat



Es tracta d'una mescla entre els dos aprenentatges anteriors, les dades d'entrenament consisteixen en exemples categoritzats i no categoritzats. Aquest cas s'utilitza quan etiquetar els exemples és costós o bé quan etiquetar-los suposa afegir un factor humà que corrompi la veracitat del mètode. En aquest cas, les dades categoritzades serveixen per identificar diferents grups de dades i prediu a quin grup pertanyen. Posteriorment, s'entrena l'algorisme amb les dades no categoritzades, d'aquesta forma es poden conèixer les fronteres que separen unes dades de les altres i també es poden definir nous grups de dades que no han estat etiquetats en les dades categoritzades.

Per exemple, en la classificació de pàgines web, es poden utilitzar pàgines web prèviament categoritzades per un expert i s'agafa més pàgines que no estan etiquetades. L'aprenentatge semi-supervisat, utilitza les primeres dades per identificar quins grups poden haver-hi i a quin

grup pertany cada pàgina. Amb les segones dades, s'acaba de definir les característiques entre les pàgines i potser identifiquen nous grups que no havien estat etiquetats per l'expert.

2.2.4. Aprenentatge reforçat

Aquesta tècnica busca les accions més adients donada una situació per tal de maximitzar una recompensa. En aquest cas, l'algorisme rep les dades per tal de descobrir un output òptim. Normalment, l'algorisme interactua amb un ambient mitjançant una seqüència d'estats i d'accions, en molts casos, l'acció a realitzar no sol afecta a la recompensa que es pugui obtenir de forma immediata, sinó que afecta als premis que es puguin obtenir amb les accions que es duran a terme en passos futurs.

Per exemple, utilitzant les tècniques d'aprenentatge reforçat, una màquina pot aprendre a jugar als escacs, la màquina ha d'aprendre, a partir de la posició actual com a input, a fer un bon moviment com a output que dirigeixi cap a la victòria. Per tal d'assolir l'objectiu la màquina juga contra una còpia d'ella mateixa.

Per tal de resumir les característiques principals dels aprenentatges esmentats anteriorment s'ha realitzat la taula 2.1.

Tipus d'aprenentatge	**Supervisat**	**No supervisat**	**Semi-supervisat**	**Reforçat**
Dades d'entrenament	Categoritzades	No categoritzades	Categoritzades i no categoritzades	No categoritzades
Output	Quantitatiu o qualitatiu	Grups de dades	Grups de dades	Següent acció
Objectiu	Predicció	Predicció	Predicció i grups de dades	Maximitzar la recompensa

Taula 2.1. Característiques principals dels tipus d'aprenentatge. (Font: Pròpia)

Per tal d'elegir el tipus d'aprenentatge que més s'adapta al plantejament del projecte, cal recordar que les dades d'entrenament de les que es disposa consisteixen en dades extretes mitjançant sensors de pressió posteriorment categoritzades, segons si pertanyen a l'usuari a identificar o no i l'objectiu serà que el programa reconegui a l'usuari quan aquest introdueixi

la contrasenya, fora del procés d'entrenament. Per tant, es tracta d'un problema de categorització sobre si la contrasenya correspon a l'usuari a reconèixer, no es correspon o bé el reconeixement és incert i, per tant, cal que l'usuari torni a reproduir el patró. Per les característiques descrites, l'aprenentatge que cal emprar és el supervisat.

2.3. Mètodes d'aprenentatge supervisat

Cada tipus d'aprenentatge presenta algorismes per tal d'obtenir el resultat desitjat. Cal recordar que en el cas de l'aprenentatge supervisat es parteix d'unes dades d'entrenament basades en uns inputs, experimentals o no, amb els outputs corresponents. D'inputs es poden tenir de qualitatius, quantitatius o bé una mescla segons el tipus de dades que s'analitzin. També es poden distingir els mateixos tipus de sortides. Les quantitatives les podem entendre com aquells outputs que poden prendre valor entre un rang de valors i tenen la característica que aquells que prenguin valors pròxims entre ells tenen característiques similars, un exemple seria la predicció de la qualitat de l'aire de Barcelona coneixent les dades històriques. En l'altre cas, si les sortides són qualitatives, prenen un valor entre un grup concret, per exemple, en el cas del reconeixement de dígits, la sortida pot prendre per valor 0, 1, 2, ..., 9. En el primer cas, la tasca de predicció amb outputs quantitatius s'anomena de regressió i, en el segon, amb outputs qualitatius, s'anomena classificació.

El tipus d'input i d'output dona lloc a un ventall molt extens d'algorismes d'aprenentatge supervisat, en aquest projecte es fa una cerca dels més significatius. Es poden distingir els algorismes de regressió, arbres de decisió, K veïns més propers, xarxes neuronals i màquines de vectors de suport.

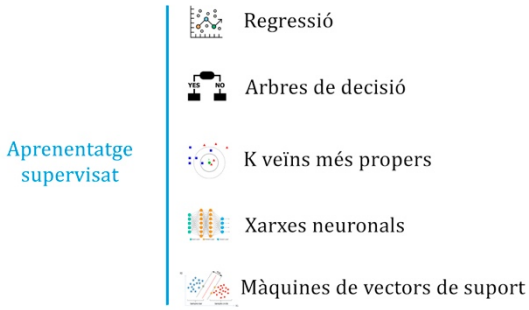


Figura 2.5. Tipus d'aprenentatge supervisat. (Font: pròpia)

2.3.1. Algorismes de regressió

Segons l'objectiu que es vulgui assolir, podem distingir dos tipus de regressió:

- **Regressió lineal.** S'utilitza quan es té un problema de regressió, és a dir, quan els outputs són quantitatius i té com a finalitat predir valors reals com número de vendes, de trucades o el cost d'habitatges. Per tal de fer la predicció, s'ajusta l'element geomètric (que depenent de les dimensions serà una línia, un pla o un hiperplà d'n dimensions) que s'adapti més als valors, aquest segueix la següent equació:

$$Y = a \cdot X + b$$

  On Y representa la variable de sortida o predicció, *a* el pendent de la l'element, X és el vector d'entrada i *b* el punt de tall amb l'eix d'ordenades. Els valors de X i Y els obtenim de les dades d'entrenament i els valors d'*a* i *b* minimitzant la distància entre els punts d'entrenament i la recta de regressió. El problema de regressió lineal s'anomena simple quan la variable X representa a una variable i múltiple quan representa a més d'una.

- **Regressió logística.** S'utilitza quan el problema és de classificació i s'utilitza per predir valors discrets, és a dir, prediu la probabilitat que es doni un estat d'entre un grup d'estats mitjançant funcions linears en X. Aquest mètode es sol emprar en casos en que el grup està format per dos estats i una sola funció lineal: 0/1, Sí/No o Verdader/Fals.

Un tipus de regressió molt utilitzada, persegueix trobar la millor corba que s'adapti a les dades, es pot utilitzar la regressió polinòmica o bé la curvilínia. MARS (*Multivariate Adaptive Regression Splines)* és un mètode de regressió en la qual diferents splines s'adapten a les dades, tal i com es pot veure en la figura 2.6.

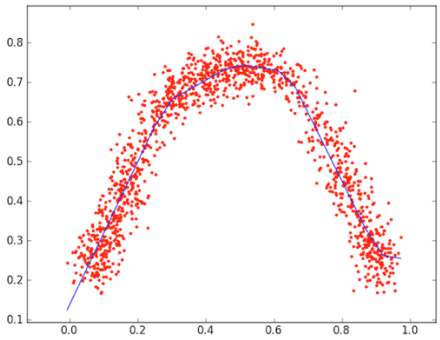


Figura 2.6. Exemple gràfic del mètode MARS. (Font: vikipedia.org)

Pel que s'ha descrit anteriorment, es reconeix aquest algorisme com una cerca de relacions entre la variable de sortida i les d'entrada.

2.3.2. Arbres de decisió

Aquest mètode es basa en dividir l'espai en grups més petits i cada grup delimitat per un rang de les variables d'entrada. El valor de la sortida serà en funció dels de valors en què es trobin les variables d'entrada. Aquest algorisme es pot implementar com a solució de problemes de regressió i de classificació, tot i que els més emprats es dediquin a resoldre els segons. Mitjançant aquest procediment és més fàcil entendre per què la màquina pren una decisió i es poden fer interpretacions de la seva decisió. Veure les figures 2.7. i 2.8. com a forma d'exemple, en aquests casos es pot veure de forma molt clara quines són les divisions o seqüències que s'han de donar per classificar una dada en un grup o l'altre.

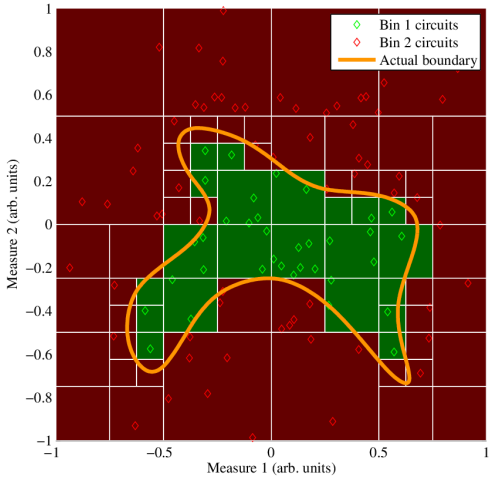


Figura 2.7. Exemple gràfic d'arbres de decisió.

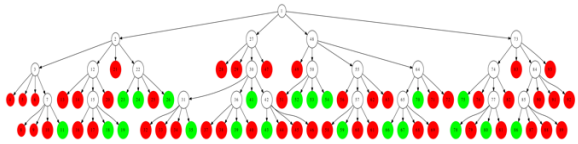


Figura 2.8. Exemple gràfic d'arbres de decisió.

(Font: A. Gómez-Pau, L. Balado, and J. Figueras, "Efficient Production Binning Using Octree Tessellation in the Alternate Measurements Space", IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 35, no. 8, pp. 1386-1395, August 2016.)

2.3.3. K veïns més propers

Aquesta tècnica no permet la fàcil interpretació dels mètodes anteriors, però és un mètode més dinàmic ja que no es parteix d'un model estructurat. També està caracteritzat per tenir una major capacitat de predicció. S'acostumen a utilitzar en problemes de classificació, tot i que si es tracta d'un problema de poques dimensions també es pot utilitzar l'algorisme per tal de resoldre problemes de regressió. La base d'aquest en problemes de classificació consisteix en torbar a quin grup depèn un punt x_0 a partir del grup al qual pertanyen els k veïns més propers que consisteixen en punts referits a les dades d'entrenament.

2.3.4. Xarxes neuronals

Aquest mètode, és similar a l'anterior, no es basa en entendre com l'algorisme arriba a una decisió, sinó que el que es valora és la precisió amb la que ho fa. Tenen un protagonisme molt important actualment ja que és l'algorisme més emprat en la intel·ligència artificial. La

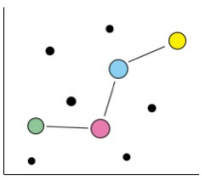
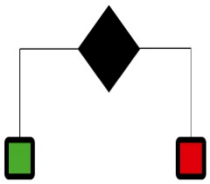
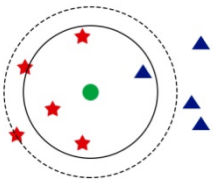
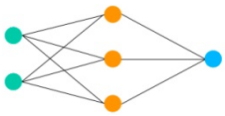
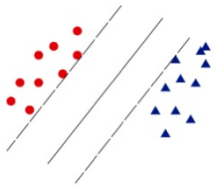
importància d'aquest mètode rau en la interconnexió entre les variables d'entrada i les de sortida, ja que permet l'obtenció de models amb un grau molt alt de complexitat a partir de neurones artificials, més endavant s'entra en detall en aquest tipus de models. Aquest algorisme pot resoldre problemes de regressió i de classificació.

2.3.5. Màquines de vectors de suport

Tot i que aquest mètode serveixi pels dos tipus de problemes, els tipus de problemes típicament resolts són de classificació. Aquest mètode es basa en dividir els punts d'entrenament mitjançant hiperplans maximitzant la separació de les classes a discernir. És capaç de treballar amb un nombre alt de dimensions. De vegades, a les dades se li apliquen funcions de transformació no lineal anomenades *Kernels* que permeten realitzar la seva separació mitjançant hiperplans.

2.4. Plantejament de la solució escollida

Com s'ha vist en l'apartat anterior, hi ha moltes alternatives per resoldre un mateix problema amb algorismes que es basen en l'aprenentatge automàtic, aquestes alternatives es recullen en la Taula 2.2. de forma resumida.

Regressió	Arbres de decisió	K veïns més propers	Xarxes Neuronals	Màquina de vectors de suport
				

Taula 2.2. Algorismes d'aprenentatge supervisat (Font: Pròpia)

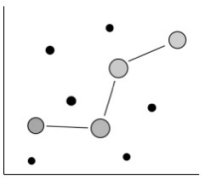
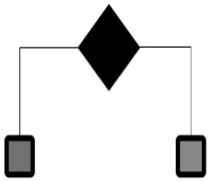
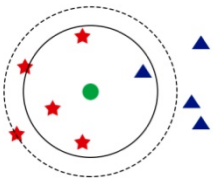
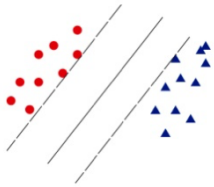
Per tal de fer una elecció correcta, cal conèixer a fons el problema que es vol solucionar i quines són les bases de partida. Per començar, les dades d'entrenament provenen d'una sèrie de sensors i s'emmagatzemen per tal que el software les tracti posteriorment, després de ser categoritzades.

En segon lloc, recordant l'objectiu del software, es busca que el programa sigui capaç de reconèixer l'usuari a identificar, un cop el programa sigui entrenat, es busca la classificació de la contrasenya segons si correspon o no a l'usuari a identificar, per tant, es tracta d'un problema de classificació. En tot cas, no importa com s'arriba a la conclusió, simplement es busca que arribi a una conclusió correcta de la forma més fiable i ràpida, es pot pensar amb l'algorisme com una caixa negra, es pot veure un diagrama que ho representa en la figura 2.9.

Figura 2.9. Diagrama de l'estructura d'un algorisme tipus k veïns més propers, xarxes neuronals o màquines de vectors de suport (Font: Pròpia)

D'aquesta forma, es descarten aquells algorismes que busquin trobar una solució mitjançant un raonament que es pugui interpretar: algorismes de regressió i arbres de decisió.

Finalment, queden les tres opcions reflectides en la taula 2.3: k veïns més propers, xarxes neuronals i màquines de vectors de suport.

Regressió	Arbres de decisió	K veïns més propers	Xarxes Neuronals	Màquina de vectors de suport
				

Taula 2.3. Opcions resultants després de fer el primer descart.

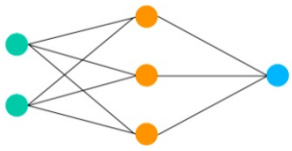
Els tres mètodes esmentats anteriorment, es tracten de tres mètodes capaços de resoldre problemes de classificació i són algorismes no-paramètrics. Que siguin no-paramètrics és important per la resolució del problema, ja que significa que el número de paràmetres dels quals depèn l'algorisme no és finit. Això els fa més complexos i amb major capacitat de predicció.

Al tractar-se de tres tècniques vàlides per la resolució, es decideix que s'empraran xarxes neuronals.

La propietat principal de les xarxes neuronals és la capacitat d'aprendre a partir d'exemples mitjançant relacions complicades i no linears entre l'entrada i la sortida [4]. Un aprenentatge d'aquest tipus permet l'adaptació a diferent tipus de soroll sense la necessitat d'entrenar l'algorisme amb tot tipus d'exemples que es puguin donar en el sistema. Una característica molt important que distingeix aquest mètode de la resta és la alta immunitat cap a patrons erronis que es poden realitzar desintencionadament quan es generen dades d'entrenament. Aquesta propietat afegeix dinamisme al software, ja que el patró de pressions no sempre serà idèntic o bé poden haver-hi errors de lectura que no han d'impedir la verificació de l'usuari.

Tot i que el procés d'entrenament de les xarxes neuronals no es caracteritza per la rapidesa, aquest model presenta una sèrie d'avantatges envers les altres alternatives destinades a funcions similars. Les xarxes neuronals generen una eina capaç d'aprendre de l'experiència i obtenir solucions a problemes que no ha entrenat mai. Per una part, les dades a partir de les quals la xarxa s'entrena poden ser poc clares, discontínues o incompletes. Per l'altra, no és necessària una definició clara del problema per tal de poder executar aquest model és prou complex per adaptar-se a tot tipus de dades.

3. Desenvolupament de la solució



La solució del problema es basarà en programar un algorisme d'aprenentatge supervisat basat en xarxes neuronals per tal de dur a terme un problema de classificació a partir de dades d'entrenament en forma de valors binaris.

3.1. Elecció del sistema

Figura 3.1. Logotip *TensorFlow*. (Font: wikipedia.org

En la programació d'intel·ligència artificial existeix una plataforma molt potent, entre d'altres, anomenada *TensorFlow*, aquesta biblioteca de codi obert va ser llançada per Google al 2015 i es basa en un sistema de xarxes neuronals. Actualment, moltes empreses empren aquesta eina per tal de dur a terme operacions diàries o bé l'utilitzen en la seva pàgina web, per exemple la companyia d'assegurances AXA utilitza el software per predir (amb un 78% d'encerts) les possibilitats que un assegurat causi un accident de cotxe [5]. Però, en la concepció d'aquest projecte, es decideix fer el programa des de zero.

Es pren aquesta decisió ja que el concepte de xarxes neuronals, com es veurà a continuació, és complex i per tal d'entendre correctament el seu funcionament calen uns coneixements previs. Per tal de reforçat la cerca sobre la teoria que regeix l'algorisme, fer el programa ajuda a tenir una visió més detallada del seu funcionament. A més, es pot adaptar el programa completament al les dades que s'utilitzen per a la resolució.

Per tal d'escriure el programa s'utilitza el llenguatge Python, ensenyat en l'Escola Tècnica Superior d'Enginyers Industrials de Barcelona on s'ha realitzat aquest projecte. També s'ha utilitzat una llibreria de Python: Numpy. Aquesta llibreria és fonamental per la programació científica amb Python ja que permet la creació de matrius i vectors, conté funcions

predefinides molt útils en el si de l'àlgebra i el càlcul. Més endavant, en l'explicació del codi es pot veure com s'ha donat ús a les possibilitats que ofereix aquesta llibreria.

Figura 3.2. Símbol de Python i Numpy, respectivament. (Font: python.org)

3.2. Introducció a les xarxes neuronals

La finalitat d'aquest apartat és demostrar que les xarxes neuronals són models eficients pel que fa al reconeixement de patrons. Per tal fi, cal conèixer les bases de funcionament. Durant aquest capítol es faran petites mencions al resultat d'escriure el codi en Python d'algunes de les funcions que regeixen el funcionament de les xarxes neuronals per tal de completar l'estructura del codi explicada en apartats posteriors.

3.2.1. Una inspiració biològica: el perceptró

La idea de les xarxes neuronals neix de la cerca d'un model matemàtic que representi un sistema biològic processant informació, per tant, que imités el comportament del cervell humà. Aquest comportament es basa en una xarxa formada per 10 bilions de neurones connectades entre elles mitjançant axons. Cada neurona rep certs impulsos electroquímics, si la suma d'aquests és suficient, permet despolaritzar la neurona, és a dir, s'activa i transmet una senyal electroquímica cap a les neurones veïnes a través de l'axó.

El primer que aconsegueix imitar aquest comportament és Frank Rosenblatt que va concebre el perceptró, inspirat pel treball previ de Warren McCulloch i Walter Pitts, els quals van idear la Neurona de McCulloch-Pitts: es tracta d'una unitat de càlcul que intenta modelar el comportament d'una neurona semblant a les que formen el cervell humà, va ser el primer model neuronal que va inspirar al desenvolupament d'altres models neuronals com el perceptró.

El perceptró és un tipus de neurona artificial que pren per entrada un conjunt de valors reals x_1, x_2, ..., i genera una sortida binària, a continuació es mostra l'esquema en la figura 3.3.

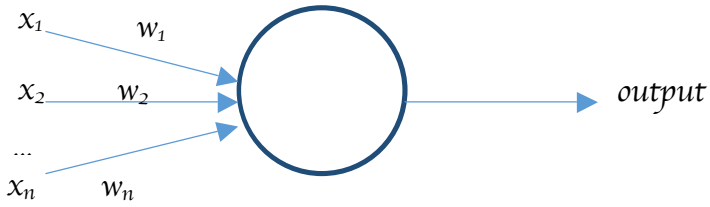


Figura 3.3. Esquema d'un perceptró. (Font: pròpia)

En aquest esquema es distingeix els valors d'entrada del perceptró, els quals es poden representar mitjançant un vector X=(x_1, x_2, ..., x_N) la connexió d'aquests amb el perceptró es fa mitjançant uns valors que es representen mitjançant els valors W=(w_1, w_2, ..., w_N) i un valor de sortida, l'output o y.

La forma en que aquestes variables es relacionen és mitjançant la següent expressió matemàtica:

$$y = f(\sum_{i=1}^{N} w_i \cdot x_i - \theta) \qquad \text{(Equació 1)}$$

On:

f és l'anomenada funció d'activació la qual segueix el següent model matemàtic:

$$f(\mu) = \begin{cases} 1, & \mu \geq 0 \\ 0, & \mu < 0 \end{cases} \qquad \text{(Equació 2)}$$

θ és el valor llindar a partir del qual el perceptró s'activa, és a dir, qual la funció d'activació val 1.

Com es pot veure aquest esquema recull la idea d'una neurona biològica com les descrites anteriorment: aquesta s'activa, prenent 1 per valor, a partir d'un valor llindar en el cas artificial i a partir d'un nombre de senyals electroquímiques en el cas biològic.

La característica principal del perceptró és que la funció de sortida pot prendre per valors 0 o 1, per tant la funció de sortida, és una funció graó, aquesta té la forma que mostra la figura 3.4.

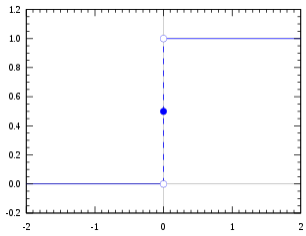


Figura 3.4. Gràfic de la funció graó (Font: wikipèdia.org)

S'observa que es tracta d'una funció no derivable ja que presenta canvis bruscos, aquesta propietat suposa un inconvenient per tal de dur a terme l'entrenament de la xarxa, tal i com es veurà més endavant.

Per tant neix la necessitat de trobar una funció que es comporti com una funció graó però sigui derivable i es conceben les neurones artificials. Aquestes segueixen la mateixa funció matemàtica que els perceptrons (equació 1) amb la diferència que la funció d'activació és derivable, un exemple són les neurones sigmoide, que utilitzen la següent funció:

$$f(\mu) = \frac{1}{1+e^{-\mu}} \qquad \text{(Equació 3)}$$

La sortida de la qual és un número entre zero i u i es tracta d'una funció derivable ja que pren la forma representada en la figura 3.5.

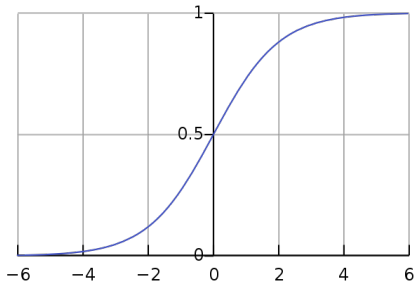


Figura 3.5. Representació gràfica de la funció sigmoide. (Font: wikipedia.org)

S'observa que les funcions d'activació són funcions no lineals. Un cop entès el funcionament d'una sola neurona, entendre el funcionament de la xarxa neuronal és més senzill.

Per tal d'implementar la funció sigmoide, es defineix la següent funció en codi Python:

```
def sigmoid(x):
    return 1 /(1 + np.exp(-x))
```

On np és l'abreviatura de la llibreria Numpy, aquest mòdul s'ha d'importar abans d'implementar la funció.

Com s'ha mencionat anteriorment, més endavant es farà ús de la derivada de la funció sigmoide, per la qual cosa es defineix la funció derivada de sigmoide amb l'equació 4.

$$f'(\mu) = \frac{e^{-\mu}}{(1+e^{-\mu})^2} \qquad \text{(Equació 4)}$$

Aquesta, té una propietat molt interessant i és que es pot escriure com a funció de la mateixa funció primitiva, és a dir, si anomenem f a la funció sigmoide i f' a la seva derivada, la funció resulta com la de l'equació 5.

$$f' = f \cdot (1 - f) \qquad \text{(Equació 5)}$$

Si es representa de la derivada de la funció sigmoide s'obté un gràfic com el recollit en la figura 3.6.

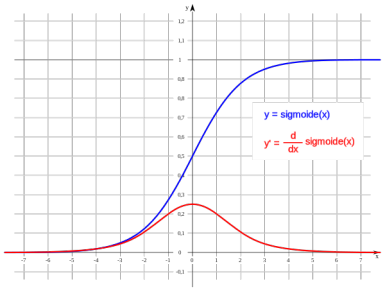


Figura 3.6. Representació gràfica de la funció sigmoide i la seva derivada. (Font: wikipedia.org)

En el codi, per simplificar, s'ha utilitzat la funció definida com en l'(Equació 4).

```
def d_sigmoid(z):
    return np.exp(-z)/(1+np.exp(-z))**2
```

Alternativament, es podria haver emprat la definició basada en la pròpia funció doncs requereix de menys càlculs.

3.2.2. Estructura d'una xarxa neuronal

L'estructura de la xarxa neuronal es basa en un conjunt de capes formades per una o més neurones artificials, en el projecte, contretament, s'han utilitzat les neurones sigmoïdals. D'aquestes capes es poden distingir tres tipus: la primera és la capa d'entrada, les neurones d'aquesta són especials ja que prenen per valor un vector d'entrada independent de la resta de neurones aquest ha d'estar entre 0 i 1 per tal d'obtenir una solució correcta, per tant, caldrà fer una modificació de les dades que es reben des dels sensors. D'altra banda es té la última capa, es caracteritza perquè el vector representa la sortida o predicció de la xarxa neuronal, per últim, tenim un conjunt de capes entre la sortida i l'entrada anomenades capes ocultes. Així com l'entrada i la sortida de la xarxa neuronal està ben definida pel problema a solucionar, un treball important a fer és decidir la quantitat de capes ocultes i de quantes neurones és cada capa, ja que aquesta decisió, en part arbitrària, afectarà al temps d'entrenament i a la precisió de la xarxa neuronal.

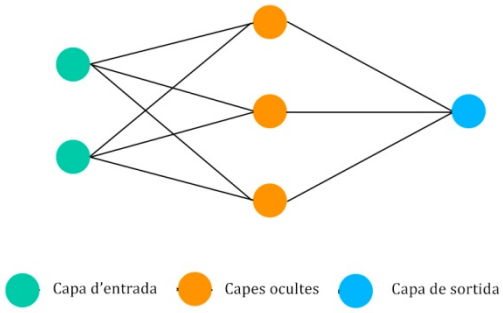


Figura 3.7. Esquema general d'una xarxa neuronal (Font: pròpia)

En la figura 3.7. es pot observar com cada neurona (representades mitjançant un cercle), excepte les d'entrada, tenen unes connexions entrants (esquerra) i unes de sortida (dreta). El valor de sortida de cada neurona és sempre el mateix tot i que la connexió vagi dirigida a diferents neurones, el que canvia el valor en l'entrada de la següent capa és el valor que prenen les connexions entre neurones.

La xarxa neuronal que s'ha desenvolupat d'aquest projecte, conté 100 neurones d'entrada, ja que cada dada d'entrenament conté 100 valors, com es veurà més endavant. Pel que fa a la capa de sortida conté una neurona que indica un valor pròxim a 0 si la contrasenya no identifica a l'usuari o 1 si sí que ho fa.

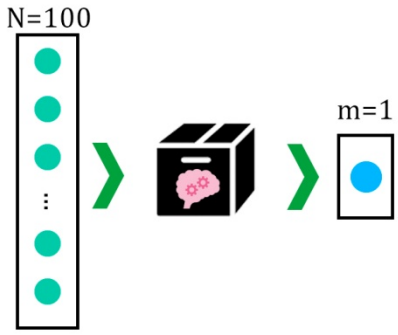


Figura 3.8. Definició entrada i sortida de l'algorisme. (Font: pròpia amb icones extretes del recurs https://icons8.com/icons)

La figura 3.8. representa l'inici de l'algorisme de les xarxes neuronals, el qual es recorda que es pot representar com una caixa negra tal i com està representat en la figura 2.7. Amb la informació que s'ha donat fins ara, es pot descobrir que dins d'aquesta caixa negra hi ha capes

ocultes de neurones artificials, cal definir el nombre de capes i neurones ocultes que entre l'entrada i la sortida de la xarxa neuronal.

L'elecció de capes ocultes és un procediment costós ja que en moltes ocasions es sol fer mitjançant prova/error i sovint la xarxa està sobre dimensionada. En el cas d'aquest projecte s'han utilitzat les següents expressions matemàtiques per tal d'obtenir una aproximació del valor òptim de neurones en dues capes ocultes [6]:

$$L_1 = \sqrt{(m+2)N} + 2\sqrt{\frac{N}{(m+2)}} \qquad \text{(Equació 6)}$$

$$L_2 = m\sqrt{\frac{N}{(m+2)}} \qquad \text{(Equació 7)}$$

On L_1 i L_2 són el número de neurones contingudes en la primera capa oculta i en la segona capa oculta, respectivament, N correspon al número de neurones en la capa d'entrada i m el de neurones en la capa de sortida.

Utilitzant les expressions: equació 6 i equació 7, s'obtenen els valors L_1=28,87 i L_2=5,77. Evidentment, els valors de les neurones han de ser enters, arrodonint a la unitat entera més propera, s'obté 29 neurones a la primera capa oculta i 6 a la segona, finalment el diagrama que representa la xarxa neuronal provisional es pot veure en la figura 3.9. Més endavant, es realitzen proves amb valors de neurones ocultes propers per veure com afecta a la solució.

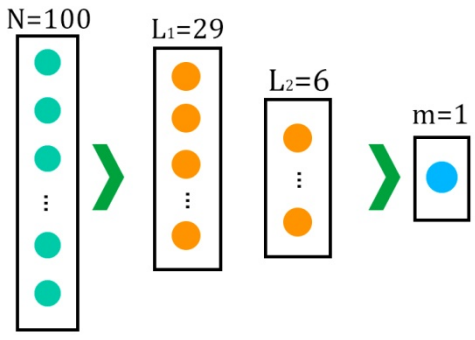


Figura 3.9. Diagrama complet de l'algorisme. (Font: pròpia)

3.2.3. Mètode d'aprenentatge

Un cop decidida l'estructura de la xarxa neuronal, comença el procés d'entrenament. Aquest consta de dues funcions: *feedforward* i *backpropagation.* Aquestes s'acostumen a representar mitjançant el sentit d'obtenció de dades, observant la figura 3.10. es pot veure com el *feedforward* avança en el sentit de la xarxa (de l'entrada a la sortida) i el *backpropagation* en sentit contrari, tot actualitzant els pesos i biaixos de la forma pertinent.

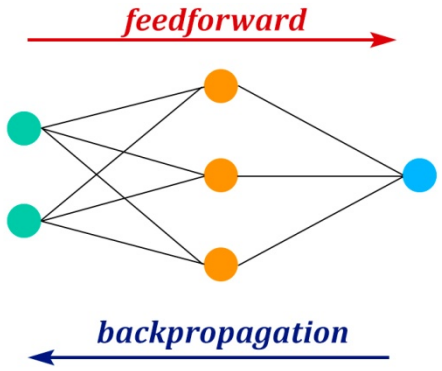


Figura 3.10. Esquema de xarxa neuronal amb processos d'aprenentatge. (Font: pròpia)

Observació: per fer més senzilla de segur l'explicació es suposa una xarxa neuronal que pren com a dades d'entrenament un vector d'entrada amb el corresponent vector de sortida, més endavant s'especifica com varien les equacions que regeix el funcionament de les xarxes neuronals davant d'un conjunt d'n dades d'entrenament.

En primer lloc, la funció *feedforward* consisteix en "alimentar" la xarxa des de la capa d'entrada fins arribar a obtenir un valor en la capa de sortida, com s'ha apreciat en la figura 3.10, aquest valor resulta ser el valor predit per la xarxa. Per fer-ho, s'utilitza una funció matemàtica molt similar a l'anunciada en el perceptró. Per tal de definir el model matemàtic, cal donar nom a les diferents parts de la xarxa. Es representa un esquema de xarxa senzilla en la figura 3.11. per tal d'entendre de forma més clara les diferents parts.

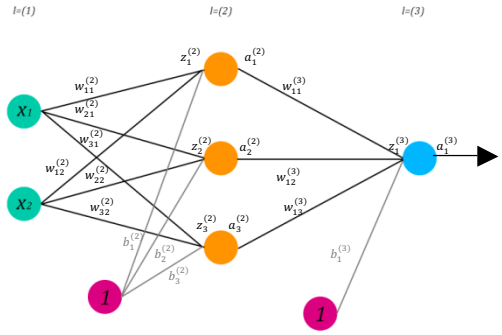


Figura 3.11. Exemple d'una xarxa neuronal amb 6 neurones i tres capes. (Font: pròpia)

En aquest exemple es pot observar una xarxa de tres capes (entrada, oculta i sortida), la capa d'entrada consta de dues neurones, l'oculta de tres i la sortida d'una. Una capa s'indica mitjançant un sobre índex $^{(l)}$ amb el número de capa, ordenat d'esquerra a dreta, entre parèntesi per tal de no confondre-ho amb potències.

Ens vincles entre neurones s'anomenen pesos es representen com $w_{ji}^{(l)}$ on el sobre índex fa referència a la capa de final de la connexió i dels subíndexs $_j$ fa referència a la neurona a la que es dirigeix el vincle i $_i$ a la neurona de la qual prové, per exemple, el pes $w_{32}^{(2)}$ fa referència al valor que pren la connexió que finalitza en la primera capa oculta i vincula la neurona 2 amb la neurona 3. Recordant el valor llindar dels perceptrons, en les xarxes neuronals es tenen els biaixos, aquests valors, representats per cercles magenta en la figura 3.11. es representen com "neurones" que prenen per valor 1 tota l'estona i el vincle entre aquestes i les neurones rep per nomenclatura $b_j^{(l)}$ on, al igual que en els pesos, $^{(l)}$ és un sobre índex que anomena la capa de final de la connexió i $_j$ fa referència a la neurona artificial a la qual afecten. Per exemple, el biaix $b_3^{(2)}$ correspon als biaixos que actuen sobre la primera capa oculta i afecta a la neurona 3.

En la figura 3.11. es poden veure les següents nomenclatures: $z_j^{(l)}$ abans de les neurones i $a_j^{(l)}$ després de les neurones. La primera, fa referència al valor que pren la neurona abans d'aplicar-li la funció d'activació, on $_j$ correspon a la neurona a la que fa referència el valor i $^{(l)}$ a la capa en que està situada la neurona. La segona, fa referència al valor d'activació de la neurona amb el mateix significat d'índexs.

Un vector d'entrada, X, pren per valors X=(x_1, x_2, ..., x_N) on N correspon al número total de neurones en la capa d'entrada, en l'exemple N=2. Un vector de sortida és Y=(y_1, y_2, ..., y_O) on O és el número de neurones de sortida, en el cas exposat O=1.

Un cop definida l'estructura i les dades de la xarxa neuronal, es passa a definir el model matemàtic que regeix el mètode de *feedforward*. En aquest, es comença a "alimentar" la xarxa des de la capa d'entrada mitjançant el vector d'entrada. Per fer-ho, s'utilitzen unes expressions matemàtiques molt semblants a les exposades en el cas del perceptró es pot apreciar en la funció (3).

$$z_j^{(2)} = \sum_{i=1}^{N} w_{ji}^{(2)} \cdot x_i + b_j^{(2)} \qquad \text{(Equació 8)}$$

$$a_j^{(2)} = f\left(\sum_{i=1}^{N} w_{ji}^{(2)} \cdot x_i + b_j^{(2)} \right) \qquad \text{(Equació 9)}$$

On $_j$ correspon a una neurona de la capa oculta, adjacent a la d'entrada, la qual es numera amb el sobre índex $^{(2)}$, $_i$ correspon a una neurona de la capa d'entrada. Si comparem la funció de l'equació 9 amb l'expressió de l'equació 1), tenen una forma molt similar exceptuant que en el cas anterior, la connexió entre l'entrada i la neurona depenia solament del número d'entrada i ara, en tenir més d'una neurona, cal distingir l'entrada i la neurona a la que es dirigeix la connexió. A més, al tenir més capes, també cal fer una distinció. També veiem un canvi en el valor llindar, que ara depèn de la neurona sobre la qual estem calculant el valor.

Pel que fa al codi Python que calcula les equacions 8 i 9, és el següent:

```
z = np.dot(w,X)+b
a = sigmoid(z)
```

On np.dot() és una funció predefinida de la llibreria Numpy que s'encarrega de la multiplicació de matrius.

Tal com indica el nom d'aquest mètode, s'alimenta la xarxa i es segueix endavant, és a dir, es calcula el valor d'activació de les neurones de la resta de capes mitjançant l'expressió de l'equació 10.

$$z_q^{(l)} = \sum_{p=1}^{P} w_{qp}^{(l)} \cdot a_p^{(l-1)} + b_q^{(l)} \qquad \text{(Equació 10)}$$

L'expressió representa el càlcul del valor d'una neurona $_q$ situada en una capa oculta o bé de sortida $^{(l)}$ a partir dels valors d'activació de les neurones $_p$ existents en la capa anterior $^{(l-1)}$ i els valors corresponents als pesos entre neurones i els biaixos.

Si es vol profunditzar més en el significat de l'equació 10 és fàcil: el valor que pren cada neurona en una capa qualsevol, depèn del valor de cada neurona de la capa anterior en funció d'un *pes* ($w_{qp}^{(l)}$) diferent per cada parella de neurones que estigui connectant i d'un *biaix* ($b_j^{(l)}$) que representa el valor llindar a partir del qual la neurona s'activa, aquesta dependència es ressalta en la figura 3.12. Cal recordar que el sentit en què s'obtenen els valors en el mètode de *feedforward* és d'esquerra a dreta.

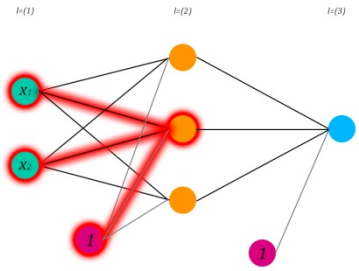


Figura 3.12. Influència de les neurones anteriors sobre una de posterior. (Font: pròpia)

L'expressió interior a la funció d'activació en l'equació 9 pot prendre per valor qualsevol real, però recordem que el que interessa és l'activació o no de la neurona $_j$, per fer-ho, s'acota el valor mitjançant una funció d'activació que escala els valors entre 0 i 1. De funcions d'activació es poden elegir entre diverses, la funció tangent hiperbòlica i la sigmoide són les més utilitzades. Les dues tenen la mateixa forma gràfica, la exposada en la figura 3.5. Degut a que la funció sigmoide està formada per exponencials i la derivada d'aquestes és més senzilla,

en aquest projecte s'utilitzarà la funció sigmoide com a funció d'activació, recordem la seva expressió:

$$\sigma(\mu) = \frac{1}{1+e^{-\mu}} \qquad \text{(Equació 11)}$$

Finalment, si es substitueix μ de l'equació 11 per l'expressió 10 la neurona $_j$ en una capa oculta o de sortida prenen el següent valor d'activació:

$$a_q^{(l)} = \sigma\left(\sum_{p=1}^{P} w_{qp}^{(l)} \cdot a_p^{(l-1)} + b_q^{(l)}\right) \qquad \text{(Equació 12)}$$

Repetint aquest pas per totes les neurones en la capa $^{(l)}$: $_{q=(1,\ldots,Q)}$, s'obté el valor d'activació de cada neurona de la capa i si es realitza per totes capes fins arribar a la de sortida obtindrem un valor d'activació en cada neurona de cada capa oculta i de la capa de sortida, el valor de la última rep el nom de predicció.

Per obtenir tots els valors d'activació de la xarxa neuronal mitjançant Python es defineix la funció *feedforward*, per fer-ho s'implementa un bucle, aquest pot ser executant un *for* o bé un *while*, en aquest cas s'han utilitzat *for* sempre que s'hagi pogut ja que consumeix menys temps:

```
for b, w in zip(self.biases, self.weights):
    z = np.dot(w,X)+b
    a = sigmoid(z)
```

On la funció *zip()* és una funció predefinida que de dues llistes, per exemple [a, b] i [x, y], en retorna els següents valors: ax i by. Els valors de z i d'a es guarden en variables, per exemple, en dues llistes (Z i A, respectivament) i s'actualitza el valor d'X per l'últim valor calculat d'a:

```
Z.append(z)
A.append(a)
X=a
```

El bucle s'executa per tots els valors de pesos i biaixos representats com *self.weights* i *self.biases,* respectivament.

L'expressió 12 permet fer una idea de la complexitat d'una xarxa neuronal: una neurona $_q$ d'una capa $^{(l)}$, depèn de tots els valors d'activació de la capa anterior $^{(l-1)}$ i, alhora, de tots els valors d'activació de les neurones de les capes anteriors. Si seguim amb aquest raonament, els valors de les neurones de l'última capa o capa de sortida, $^{(L)}$, depèn dels valors d'activació de la capa anterior $^{(L-1)}$ i dels valors d'activació de totes les neurones de les capes anteriors, veure dependència en la figura 3.13. Donant lloc a una funció de sortida no-linear i extremadament complexa.

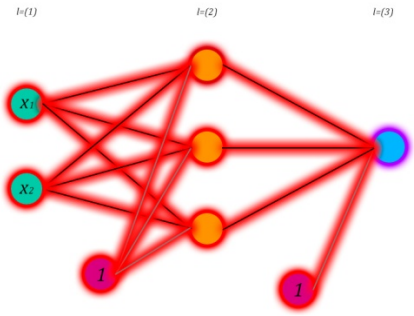


Figura 3.13. Influència de totes les neurones de la xarxa sobre la sortida. (Font: Pròpia)

Un cop el mètode de *feedforward* arriba a calcular els valors d'activació en l'extrem de la sortida valor de la neurona ressaltada amb lila en la figura 3.13, es dona la tècnica de *backpropagation*, aquesta, es basa en comparar el valor predit a la sortida de la xarxa neuronal amb el valor desitjat tal i com es veu en la figura 3.14, i ajustar-ne el valor fent petites modificacions als pesos i als biaixos.

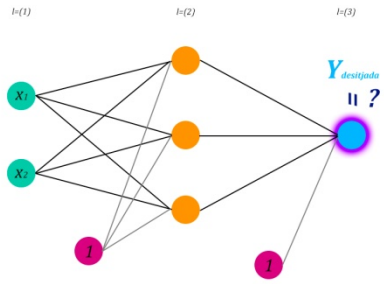


Figura 3.14. Comparació del valor de la neurona de sortida amb la sortida desitjada. (Font: pròpia)

A continuació, s'explica el mètode de *backpropagation* amb més detall. Cal recordar que les dades d'entrenament estan formades vectors d'entrada i les seves sortides desitjades corresponents a cada vector d'input.

Del *feedforward*, s'obtenen els valors $a_q^{(L)}$, on $_q$ correspon a una neurona q de la capa de sortida $^{(L)}$, per tant, un valor $a_q^{(L)}$ per cada neurona de sortida. Aquesta predicció, es compara amb el vector de sortida desitjat, Y=(y_1, y_2, ..., y_q), que assignen les dades d'entrenament. Per fer-ho s'utilitza la funció de cost, aquesta ens dona una idea de la diferència entre el valor predit i el desitjat d'aquesta forma es pot conèixer si la predicció està molt lluny de la realitat o bé ha estat força correcta:

$$E = \frac{1}{2}(a_q^{(L)} - y_q)^2 \qquad \text{(Equació 13)}$$

Aquesta funció d'error rep el nom de funció de suma de quadrats o funció de cost que quantifica el nivell d'encert de la xarxa neuronal

Per calcular el valor de l'error mitjançant Python, cal recordar la matriu A, on es guarden tots els valors d'activació de totes les neurones d'activació i recuperar el vector de sortides desitjades i s'executa:

E = (1/2)(A[-1] - target)**2*

L'objectiu es reduir al mínim E, fins que el valor predit per la xarxa neuronal sigui prou pròxim al valor desitjat. Per tal d'assolir aquest objectiu, tal com s'ha comentat anteriorment,

es fan petites modificacions als pesos i als biaixos de tal forma que E sigui pròxima a 0, aquestes modificacions es fan de la següent manera:

$$w^{(\tau+1)} = w^{(\tau)} + \Delta w^{(\tau)} \quad \text{(Equació 14)}$$

$$b^{(\tau+1)} = b^{(\tau)} + \Delta b^{(\tau)} \quad \text{(Equació 15)}$$

On τ simbolitza la iteració de *backpropagations* feta.

Per dur a terme la funció d'optimització de la funció cost (equació 13), s'utilitza el descens pel gradient, aquest mètode d'optimització consisteix en fer les següents modificacions:

$$w^{(\tau+1)} = w^{(\tau)} - \eta \nabla E(w) \quad \text{(Equació 16)}$$

$$b^{(\tau+1)} = b^{(\tau)} - \eta \nabla E(b) \quad \text{(Equació 17)}$$

On η > 0 és el coeficient d'aprenentatge del qual es parla al final d'aquest capítol.

S'observa que al enunciar les funcions anteriors (equació 16 i equació 17), s'especifica que es calcula el gradient de la funció cost respecte els pesos i els biaixos, respectivament, però si observem la funció 13 la suma de quadrats no depèn d'aquests. Per tal de calcular les variacions del cost respecte tots els pesos i els biaixos, es fa servei de la regla de la cadena.

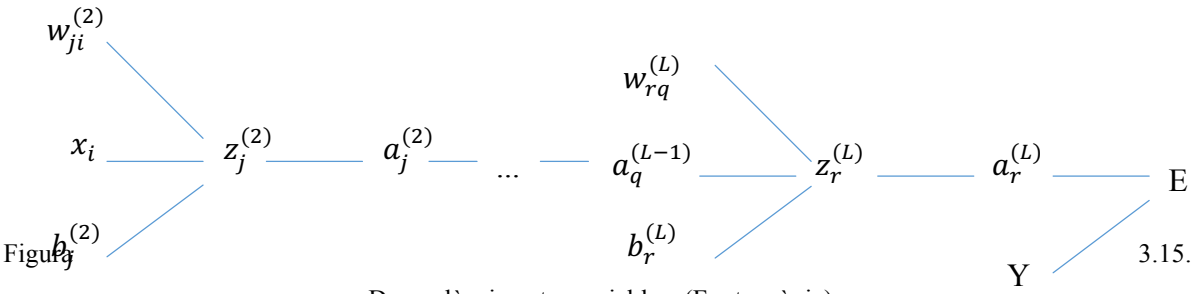


Figura 3.15. Dependència entre variables. (Font: pròpia)

Mitjançant l'esquema de la figura 3.15. és més senzill definir la regla de la cadena. Per començar, es defineix com varia el cost segons un pes qualsevol, per exemple, com varia E respecte el pes $w_{qp}^{(l)}$:

$$\frac{\partial E}{\partial w_{qp}^{(l)}} = \frac{\partial E}{\partial z_q^{(l)}} \cdot \frac{\partial z_q^{(l)}}{\partial w_{qp}^{(l)}} \qquad \text{(Equació 18)}$$

En l'(Equació 10) es pot comprovar, que $z_q^{(l)}$ depèn directament de $w_{qp}^{(l)}$, també es pot verificar a través de la figura 3.15. en la qual el valor $z_r^{(L)}$ està connectat directament amb $w_{rq}^{(L)}$, però E no depèn directament de $z_q^{(l)}$. Es tornar a aplicar la regla de la cadena:

$$\frac{\partial E}{\partial z_q^{(l)}} = \frac{\partial E}{\partial z_r^{(l+1)}} \cdot \frac{\partial z_r^{(l+1)}}{\partial z_q^{(l)}} \qquad \text{(Equació 19)}$$

On r representa una neurona qualsevol en la capa immediatament després de la capa de neurones q. En el cas representat en l'esquema de la figura 3.15. $(L)=(l+1)$ i com $z_r^{(L)}$ està connectada directament amb E a través d'$a_r^{(L)}$ s'hauria arribat al final, del contrari, es repetiria el procés fins arribar al cas descrit anteriorment en que es calcula la variació del cost respecte els valors de les neurones abans d'aplicar la funció d'activació.

Si s'aplica l'equació 19 a l'equació 18, s'obté:

$$\frac{\partial E}{\partial w_{qp}^{(l)}} = \frac{\partial E}{\partial z_r^{(l+1)}} \cdot \frac{\partial z_r^{(l+1)}}{\partial z_q^{(l)}} \cdot \frac{\partial z_q^{(l)}}{\partial w_{qp}^{(l)}} \qquad \text{(Equació 20)}$$

A continuació es defineix una notació important per simplificar l'expressió (Equació 20):

$$\delta_q^{(l)} \equiv \frac{\partial E}{\partial z_q^{(l)}} \qquad \text{(Equació 21)}$$

També podem fer les següents simplificacions aplicant les equacions 10, 12 i la regla de la cadena:

$$\frac{\partial z_q^{(l)}}{\partial w_{qp}^{(l)}} = a_p^{(l-1)} \qquad \text{(Equació 22)}$$

$$\frac{\partial z_r^{(l+1)}}{\partial z_q^{(l)}} = w_{rq}^{(l+1)} \cdot \sigma'(z_p^{(l)}) \qquad \text{(Equació 23)}$$

Si a l'equació 20 se li aplica les condicions anteriors (equació 21, equació 22 i equació 23), resulta:

$$\frac{\partial E}{\partial w_{qp}^{(l)}} = \delta_q^{(l)} \cdot a_p^{(l-1)} \qquad \text{(Equació 24)}$$

On $\delta_q^{(l)}$ segons l'equació 20, 21 i 23:

$$\delta_q^{(l)} \equiv \frac{\partial E}{\partial z_q^{(l)}} = \sum_{q=1}^{Q} \delta_p^{(l+1)} w_{rq}^{(l+1)} \cdot \sigma'\left(z_q^{(l)}\right) \qquad \text{(Equació 25)}$$

On $_q$ representa una neurona en la capa $^{(l)}$ que té un total de $_Q$ neurones.

Per part dels biaixos es segueix una metodologia molt similar de forma que s'obté la següent equació:

$$\frac{\partial E}{\partial b_q^{(l)}} = \delta_q^{(l)} \qquad \text{(Equació 26)}$$

Per calcular totes les deltes de totes les neurones de la xarxa, es comença per calcular la delta corresponent a la neurona de sortida, per fer-ho mitjançant codi Python, s'utilitza la següent funció:

*delta_L = (A[-1] - target) * d_sigmoid(z_L)*

Un cop calculada la primera, seguim calculant la resta en el sentit que marca el mètode de *backpropagation* fins arribar a les neurones d'entrada, que d'aquestes no se'n calculen les deltes ja que no presenten variacions en el seu valor.

*delta_l = (np.dot((self.weights[i+1]).T, delta_ant))*d_sigmoid(z)*

Per tal d'entendre-ho millor, es recupera l'exemple de la figura 3.11. , primer de tot, cal recordar la nomenclatura, en la capa d'entrada $^{(1)}$ les neurones s'anomenen $_{i=(1,\dots,N)}$ on N=2, en la capa oculta $^{(2)}$ les neurones s'anomenen $_{j=(1,\dots,M)}$ on M=3 i en la capa de sortida $^{(3)}$ les neurones s'anomenen $_{k=(1,\dots,O)}$ on O=1. Els pesos que relacionen les capes $^{(2)}$ i $^{(1)}$ prenen per nomenclatura $w_{ji}^{(2)}$ i els que ho fan per les capes $^{(3)}$ i $^{(2)}$ $w_{kj}^{(3)}$. Per últim, els biaixos que afecten els valors d'activació de la capa $^{(2)}$ i $^{(3)}$, s'anomenen $b_j^{(2)}$ i $b_k^{(3)}$, respectivament.

En primer lloc, calculem les variacions respecte als pesos començant per com varia el cost respecte els pesos més pròxims, aquests són $w_{kj}^{(3)}$:

$$\frac{\partial E}{\partial w_{kj}^{(3)}} = \delta_k^{(3)} \cdot a_j^{(2)}$$

On $\delta_k^{(3)} \equiv \frac{\partial E}{\partial z_k^{(3)}} = (a_k^{(3)} - y_k) \cdot \sigma'(z_k^{(3)})$

Per tant,

$$\frac{\partial E}{\partial w_{kj}^{(2)}} = \sum_{k=1}^{O} \left(a_k^{(3)} - y_k\right) \cdot \sigma'\left(z_k^{(3)}\right) \cdot a_j^{(2)}$$

L'equació anterior s'aplica per cada $_{j=(1,2,3)}$ de forma que obtenim tres valors, un per cada pes.

Ídem amb els biaixos $b_k^{(3)}$,

$$\frac{\partial E}{\partial b_k^{(3)}} = \sum_{k=1}^{O} \left(a_k^{(3)} - y_k\right) \cdot \sigma'\left(z_k^{(3)}\right)$$

En aquest cas sol hi ha un biaix, per tant, obtindrem un valor.

Si ara es reprodueix el mateix procediment pels pesos i biaixos de l'inici, obtenim:

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = \delta_j^{(2)} \cdot x_i$$

On $\delta_j^{(2)} \equiv \frac{\partial E}{\partial z_j^{(2)}} = \sum_{k=1}^{O} \delta_k^{(3)} w_{kj}^{(3)} \cdot \sigma'\left(z_j^{(2)}\right)$

Per tant,

$$\frac{\partial E}{\partial w_{ji}^{(2)}} = \sum_{k=1}^{O} \delta_k^{(3)} w_{kj}^{(3)} \cdot \sigma'\left(z_j^{(2)}\right) \cdot x_i$$

S'observa que en aquest cas, les neurones de la capa $^{(1)}$ són les neurones d'entrada. En aquest cas, s'obtenen 6 valors, un per cada connexió entre neurones.

Ídem amb els biaixos,

$$\frac{\partial E}{\partial b_j^{(2)}} = \sum_{k=1}^{O} \delta_k^{(3)} w_{kj}^{(3)} \cdot \sigma'\left(z_j^{(2)}\right)$$

S'obté tres valors de biaixos.

Finalment, es pot concloure:

$$\nabla E(w) = \begin{pmatrix} \frac{\partial E}{\partial w_{ji}^{(2)}} \\ \frac{\partial E}{\partial w_{kj}^{(3)}} \end{pmatrix} i \ \nabla E(b) = \begin{pmatrix} \frac{\partial E}{\partial b_j^{(2)}} \\ \frac{\partial E}{\partial b_k^{(3)}} \end{pmatrix}$$

Un cop obtinguts aquests valors, per tal de resoldre les equacions 16 i 17, cal donar valor al coeficient d'aprenentatge, η.

Per fer-ho, s'explica el funcionament del descens pel gradient de forma gràfica. Per tal de simplificar l'explicació, es suposa una funció convexa en dues dimensions com la següent:

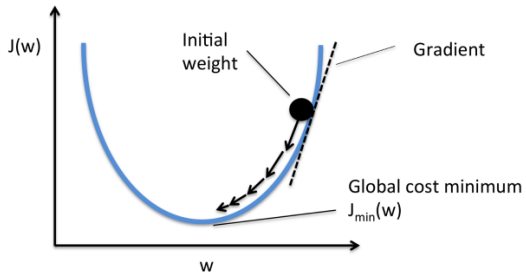


Figura 3.16. Funció convexa de dues dimensions. (Font: wikipedia.org)

L'objectiu de l'algorisme del descens pel gradient consisteix en trobar el mínim d'una funció. Per fer-ho, es parteix d'un punt aleatori, en la figura 3.16. l'anomenat *Initial weight*, i es realitzen petits passos en sentit contrari al del gradient, ja que l'objectiu es minimitzar. Aquests passos que es prenen poden ser constants o adaptar-se de forma que si s'està lluny del mínim es prenen passos més ràpid i cada cop que el nou punt s'acosta al mínim els passos són més petits per tal de tenir millor resolució del punt en què es troba el mínim. Aquest mètode, s'utilitza per evitar el problema en què per fer un pas massa gran la funció no convergeixi ja que és incapaç de trobar el mínim, aquest problema s'il·lustra de la següent forma:

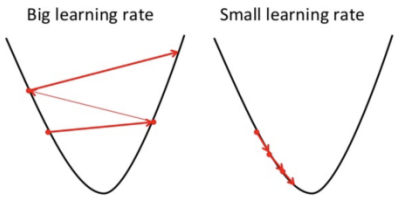


Figura 3.17. Representació del coeficient d'aprenentatge. (Font: wikipedia.org)

Un cop explicat el problema amb dos dimensions és més fàcil entendre el problema al aplicar el descens pel gradient en les xarxes neuronals, on tenim tantes dimensions com pesos i biaixos es tinguin, en l'exemple, 13 dimensions. En el cas de les xarxes neuronals, per tant, la funció a minimitzar és una funció no convexa, és a dir, pot tenir diversos mínims locals en un espai n-dimensional. Tot i tenir aquestes condicions, el procediment és el mateix: es parteix d'un punt aleatori, és a dir, una valors de pesos i biaixos establerts de forma aleatòria i es realitzen petits passos, de valor el coeficient d'aprenentatge en direcció el mínim local.

Aquest coeficient d'aprenentatge, al igual que en el cas 2-dimensional es pot adaptar en funció de la proximitat al mínim local, aquesta és una forma d'assegurar trobar un mínim prou fiable minimitzant el temps d'operació.

Fins aquí el procés de *backpropagation,* un cop enunciats ambdós processos, cal saber com relacionar-los per tal d'arribar a l'òptim.

Per tal que la xarxa neuronal sigui capaç "d'aprendre" i assoleixi el seu objectiu, que en aquest cas consisteix en fer una classificació d'una contrasenya en vàlida o errònia, cal seguir el següent procediment:

Per començar, com ja s'ha comentat, es defineix un punt aleatori de partida de la xarxa, aquest punt es troba donant valor a tots els pesos i biaixos de la xarxa. Un cop definit, s'introdueixen els valors experimentals d'entrada i mitjançant les equacions de *feedforward* s'obtenen tots els valors d'activació de la xarxa neuronal s'arriba al valor que realment importa, aquest és el valor d'activació a la sortida de la xarxa. Aquest es compara amb el valor que hauria de tenir la sortida, aquest valor s'obté de les dades d'entrenament i la comparació es fa mitjançant una funció de cost que ens dona una idea de l'error que comet la xarxa en predir el valor. Aquest error dóna lloc a la delta, que s'ha vist que forma part de la correcció a fer als paràmetres establerts inicialment. Amb aquest error i el mètode de *backpropagation* es propaga cap a capes anteriors a la de sortida fins que arriba a la capa d'entrada. A través del procés de *backpropagation* s'obté una correcció dels pesos i els biaixos, aquesta s'aplica i es torna a començar el procés amb els nous valors. Aquest procediment es repeteix en bucle amb els mateixos valors d'entrada fins que la xarxa retorni una sortida similar a l'esperada o bé fins que la funció cost arribi al mínim local i els canvis en la sortida siguin mínims o nuls. A continuació, en la figura 3.18. es pot veure un diagrama del bucle principal en què es basa l'algorisme de les xarxes neuronals.

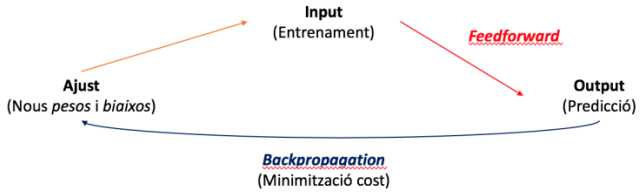


Figura 3.18. Bucle principal de l'algorisme. (Font: Pròpia)

3.3. Dades d'entrenament

Tota l'explicació continguda els apartats anteriors respecte les xarxes neuronals, s'ha fet sobre el supòsit que sol s'entrena sobre un vector d'entrada amb la sortida corresponent. Com s'ha explicat, aquest entrenament no s'escau amb la realitat, ja que es necessita un conjunt de dades d'entrenament. En aquesta part, es detalla aquesta informació, ja que és útil per tal d'entendre la forma del codi.

En aquest apartat s'interacciona amb el hardware, corresponent al treball mencionat anteriorment, mitjançant la transmissió de dades hardware-software. Aquestes dades arriben amb un format particular, és interessant entrar-hi en detall per tal d'entendre el tractament d'aquestes dades per part del software.

Per tal d'entrenar la xarxa per que pugui reconèixer l'usuari a identificar, fan falta moltes dades d'entrenament i no sol de l'usuari a identificar, sinó moltes altres contrasenyes que no l'identifiquin. Un alt nombre de contrasenyes a entrenar ajuda a la xarxa a reconèixer amb més fidelitat a l'usuari.

De dades en distingim de dos tipus:

3.3.1. Dades d'entrenament

De cada mostra extreta del hardware, s'obté un conjunt de 202 valors, el primer correspon a un byte fix que anuncia l'arribada de dades, els 200 següents consisteixen en 100 dades binàries dividides en dues parts, cada dada formada per valors de 16 bits, d'aquests, els 15 últims formen part de la contrasenya de l'usuari i el primer és un dígit extra que val sempre 0

fins que no s'enviï a última dada de les 200. L'últim valor de les dades és un valor de comprovació, una suma directa de tots els valors rebuts per tal de comprovar que les dades s'han rebut correctament i no hi ha un problema de transmissió.

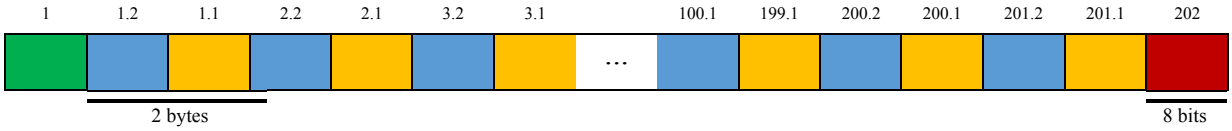


Figura 3.19. Seqüència de valors rebuts des del hardware. (Font: pròpia)

En la figura 3.19. es representen les dades rebudes des del hardware, per ordre (d'esquerra a dreta) es pot apreciar el primer valor (1) que representa un valor de 8 bits que indica l'inici de l'enviament de dades, en el codi de colors blau i groc es representen els valors del patró de pressions separats per dues parts, primer s'envia la segona i després la primera. Per últim, la seqüència acaba amb el valor vermell (202) que torna la suma directa en binari dels 200 valors enviats anteriorment.

Aquestes dades s'obtenen a través dels sensors però es guarda un fitxer amb totes les contrasenyes, de 100 valors, que es posseeixin: de l'usuari a identificar i de la resta d'usuaris. De forma que cada xarxa estarà entrenada per identificar un sol usuari. Per cada valor dels 100 se li assigna un bit més com a bit de sortida, aquest, identificarà si es tracta de l'usuari a identificar (1) o bé es tracta d'un altre usuari (0).

3.3.2. Dades de l'usuari a identificar

Un cop la xarxa estigui entrenada, l'usuari desitja identificar-se, en aquest moment s'avalua la xarxa, és a dir, es fa el procediment del *feedforward* amb el valor d'entrada, ara provinent dels sensors.

Es recorda que la forma de cada contrasenya és la mateixa: un conjunt de 100 valors. Aquests valors, es recorda, que estan formats per valors binaris formats per dues parts. Aquesta estructura s'obté directament dels sensors. Però com s'ha mencionat en l'estructura de les xarxes neuronals, la capa d'entrada accepta valors entre 0 i 1, així que es modifiquen les dades per a que els valors d'entrada siguin els correctes, per fer-ho, s'uneixen les parts dels valors binaris i el número de 16 bits es transforma en decimal. S'obtenen nombres enters entre

el 0 i el 65.535 valors corresponents entre 16 bits amb valor 0 i 16 amb valor 1, finalment, cal escalar aquests valors.

Per fer l'escalat, s'estableix el valor més petit que es pot rebre 0 i és assignat al valor més petit que poden prendre les dades escalades: 0 i es repeteix el procés en el cas del valor màxim, 65.535 passa a ser 1. Per situar un nombre extret del sensor i passat a decimal, s'utilitza la següent expressió:

$$esc = \frac{valor - mínim}{màxim - mínim} \qquad \text{(Equació 27)}$$

On *valor* és el valor binari de 16 bits transformats a decimal, *mínim* correspon al valor més petit que pot prendre *valor*, 0 i *màxim* el valor més gran, 65.535.

3.5. Comunicació de les dades

Pel que fa a la comunicació de les dades es planteja fer la comunicació de les dades del hardware fins al software de forma xifrada. D'aquesta forma, es garanteix un sistema d'identificació segur i difícil de desxifrar.

Així, el software ha de ser capaç de desxifrar les dades per tal d'obtenir els valors de les dades abans de ser xifrades. Per fer aquest procediment, es mira d'implementar l'RSA (Rivest-Shemir-Adelman).

Aquest algorisme és de xifratge de clau pública, és a dir, la clau utilitzada per xifrar les dades i la clau per desxifrar-les són diferents. L'usuari encarregat de xifrar les dades utilitza dues claus: una de privada i una de pública que comparteix amb el receptor.

A continuació s'explica en detall el funcionament de l'algorisme, tot i que en aquest treball sol s'utilitza la part de desxifrat, s'explica també el primer pas, el de xifratge, per tal d'entendre el funcionament en conjunt.

3.5.1. Xifrar

Es comença per l'elecció d'un parell de nombres de xifratge, aquests han de ser primers (c_1, c_2) (p. e. (3, 11)). Es calcules els següents valors:

$$n = c_1 \cdot c_2 \quad \text{(Equació 28)}$$

$$\emptyset = (c_1 - 1) \cdot (c_2 - 1) \quad \text{(Equació 29)}$$

Pels valors de l'exemple s'obté $n = 33$ i $\emptyset = 20$. Ara, es selecciona un número, *k*, imparell i que no tingui múltiples amb $\emptyset$, per exemple $k = 7$. D'aquesta forma queda definida la clau pública *(n, k)*, en l'exemple *(33, 7)*.

Un cop trobada la clau pública, es busca la clau privada, per fer-ho cal fer servei de l'equació 30.

$$k \cdot j = 1 \ (mod \ \emptyset) \quad \text{(Equació 30)}$$

On *j* representa la clau privada. Per tant, s'ha de trobar un valor j tal que el residu de dividir $\frac{k \cdot j}{\emptyset}$ ha de ser zero. En l'exemple, $j = 3$.

A continuació es transformen les dades, si s'escau, en nombres, en el cas que tracta aquest projecte, s'ha de passar de nombres binaris (p. e. 00000010) a decimals (per l'exemple, 2). Un cop feta la transformació de les dades, es repeteix per cada dada a xifrar l'Equació 31.

$$a^k = a' \ mod \ n \quad \text{(Equació 31)}$$

On a' correspon al valor xifrat d'a. En l'exemple $2^7 = a' mod \ 33$. Si es calcula el residu de $\frac{a^k}{n}$ resulta 29. Si transformem el número 29 decimal a binari expressat en una variable de 8 bits, s'obté 00011101. Aquest valor és el text xifrat.

3.5.2. Desxifrar

Un cop clara la forma de xifrar un valor se'n fa l'operació inversa. Per fer-ho es necessita conèixer els mateixos valors que s'han utilitzat en la part del xifratge i es parteix del valor que es rep, per exemple, 00011101. Primer es transforma a decimal, en l'exemple, 29. En segon lloc, es fa servei de l'equació 32.

$$a'^{j} = a \; mod \; n \qquad \text{(Equació 32)}$$

Si s'aplica la funció 26 amb els valors d'exemple s'obté el següent: $29^3 = a \; mod \; 33$, si es resol l'equació s'obté $a = 2$, si representem a en binari s'obté 00000010, que correspon al valor abans de ser xifrat.

3.4. Tractament de les dades per la xarxa neuronal

Un cop conegudes les dades d'entrenament, es pot especificar més pel que fa a la forma de les dades que seran tractades per la xarxa neuronal i entraran en els processos de *feedforward* i *backpropagation*.

Cada contrasenya vindrà donada per 100 valors, per tant, la xarxa tindrà 100 neurones d'entrada i la classificació de les contrasenyes ve donada per un valor de sortida, segons si aquesta correspon a l'usuari a identificar o no. Per tant, es tindrà un a neurona en la capa de sortida.

La forma de tractar les dades és la següent, cada contrasenya es representa mitjançant una columna en una matriu, X, de 100 files i n columnes on $_n$ és el nombre total de dades d'entrenament. Pel que fa als valors de sortida, Y, seran una matriu de una fila, ja que es té una neurona en l'última capa i d'n columnes, una per cada contrasenya d'entrenament.

$$X = \begin{pmatrix} x_{1,1} & \cdots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{100,1} & \cdots & x_{100,n} \end{pmatrix}, Y = (y_{11} \quad \cdots \quad y_{1n})$$

Pel que fa als pesos, també tenen forma matricial, si recordem l'expressió d'un pes que connecta una neurona $_p$ de la capa $^{(l)}$, amb una neurona q de la capa $^{(l+1)}$ és $w_{qp}^{(l)}$, aquesta notació no és arbitrària, sinó que serveix per classificar els pesos en una matriu de tantes files com neurones hi ha en la capa $^{(l+1)}$, Q, i tantes columnes com neurones hi ha a la capa $^{(l)}$, P, aquesta matriu té les següents dimensions (QxP).

Els biaixos a diferència dels pesos tenen una forma vectorial, la notació utilitzada és $b_q^{(l)}$. El vector emprat per definir els biaixos que afecten a les expressions matemàtiques de la capa $^{(l)}$ té una llargada de Q.

$$w^{(l)} = \begin{pmatrix} w_{11}^{(l)} & \cdots & w_{1P}^{(l)} \\ \vdots & \ddots & \vdots \\ w_{Q1}^{(l)} & \cdots & w_{QP}^{(l)} \end{pmatrix}, b^{(l)} = \begin{pmatrix} b_1^{(l)} \\ \vdots \\ b_Q^{(l)} \end{pmatrix}$$

Ara es pot concretar quina és la forma de les expressions matemàtiques del mètode de *feedforward*. Pel que fa a l'equació 9 es pot escriure de la següent forma matricial:

$$a^{(l+1)} = \sigma\left(w^{(l)} \cdot X + b^{(l)}\right) \tag{33}$$

Per tant, la multiplicació entre els pesos i els valors d'entrada és matricial i la funció sigmoide s'aplica a cada valor de la matriu. Mitjançant aquest tipus de multiplicació i el correcte posicionament de les matrius, s'assoleix una matriu amb dimensions (Qxn), per tant, s'obté una matriu on cada columna conté els valors d'activació de totes les neurones de la capa $^{(l+1)}$.

Seguint aquest procediment, s'obté una matriu molt similar a l'anterior però en aquest cas la matriu té tantes files com neurones a la capa de sortida $^{(L)}$, en el cas de la xarxa que s'implementa en aquest projecte és tracta d'una fila i tantes columnes com dades d'entrenament, n.

$$a^{(L)} = \begin{pmatrix} a_{11}^{(L)} & \ldots & a_{1n}^{(L)} \end{pmatrix}$$

Un cop s'obté el valor de l'activació en les neurones de sortida, comença la part de *backpropagation*. L'objectiu d'aquesta, és trobar les modificacions que cal fer als pesos i als biaixos per tal de reduir la funció cost (equació 13), però ara no es té una sola dada d'entrenament, sinó que se'n disposen n, el que provoca una modificació de l'equació 13 per tal de calcular la funció cost de totes les dades que s'entrenen:

$$C(w,b) = \frac{1}{2}\sum_n (a^{(L)} - Y)^2 \qquad \text{(Equació 34)}$$

En alguns casos, es pot trobar que la fracció ½ varia per 1/(2n) ja que es fa per n mostres, en tot cas, aquest canvi no és significatiu alhora d'implementar les xarxes neuronals, així que s'utilitza l'equació 34 per tal de calcular l'error.

Seguint amb el backpropagation, es vol trobar les variacions a realitzar als pesos i als biaixos, recordant les equacions 24 i 26, es poden escriure les següents formes matricials:

En general,

$$\frac{\partial C}{\partial w^{(l)}} = \delta^{(l)} \cdot a^{(l-1)} \qquad \text{(Equació 35)}$$

$$\frac{\partial C}{\partial b^{(l)}} = \delta^{(l)} \qquad \text{(Equació 36)}$$

Ja s'ha vist anteriorment com canvia el càlcul de la delta segons la capa on s'estigui calculant, de forma matricial es distingeixen els mateixos casos:

Si es tracta de l'última capa,

$$\delta^{(L)} = \left(a^{(L)} - Y\right) \odot \sigma'\left(z^{(L)}\right) \qquad \text{(Equació 37)}$$

En aquesta equació apareix el producte Hadamard, ⊙, aquest simbolitza el producte de matrius element a element. La dimensió de la matriu obtinguda és (Oxn), és a dir, tantes files com el número de neurones de sortida i tantes columnes com dades d'entrenament.

Si es fa el càlcul per una capa oculta qualsevol,

$$\delta^{(l)} = (w^{(l)})^T \cdot \delta^{(l+1)} \odot \sigma'(z^l) \quad \text{(Equació 38)}$$

La dimensió de delta és tantes files com neurones hi hagi en la capa $^{(l)}$ i tantes columnes com dades d'entrenament.

Resumint, en el *backpropagation* es tenen les següents equacions:

$$\frac{\partial C}{\partial w^{(l)}} = ((w^{(l)})^T \cdot \delta^{(l+1)} \odot \sigma'(z^l)) \cdot a^{(l-1)} \quad \text{(Equació 39)}$$

$$\frac{\partial C}{\partial w^{(L)}} = \left(\left(a^{(L)} - Y\right) \odot \sigma'\left(z^{(L)}\right)\right) \cdot a^{(l-1)} \quad \text{(Equació 40)}$$

$$\frac{\partial C}{\partial b^{(l)}} = (w^{(l)})^T \cdot \delta^{(l+1)} \odot \sigma'(z^l) \quad \text{(Equació 41)}$$

$$\frac{\partial C}{\partial b^{(L)}} = \left(a^{(L)} - Y\right) \odot \sigma'\left(z^{(L)}\right) \quad \text{(Equació 42)}$$

Per últim s'utilitza les equacions 16 i 17 per renovar els pesos i els biaixos i es torna a començar l'algorisme.

Finalment, al igual que en el cas d'una sola dada d'entrenament, es realitzen aquests passos de forma reiterada fins obtenir el mínim en la funció de cost.

3.5. Codi

Un cop entès el funcionament de les xarxes neuronals, a continuació s'entra en detall en el codi que s'ha utilitzat. En aquest apartat no es troba el codi sencer sinó que és una guia dels elements bàsics que formen el codi, aquest es pot trobar complet a l'Annex A al final d'aquesta memòria.

En la figura 3.20. es pot apreciar un esquema que mostra l'estructura del software i les funcions que el formen.

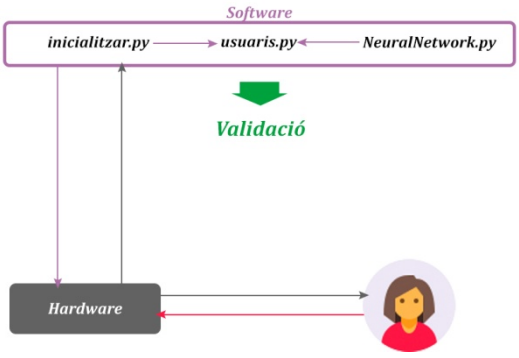


Figura 3.20. Esquema codi. (Font: Pròpia amb icones extretes del recurs https://icons8.com/icons)

Pel que fa al funcionament intern del software i la jerarquia de les funcions, en la figura 3.21. en podem veure un esquema que ho clarifica.

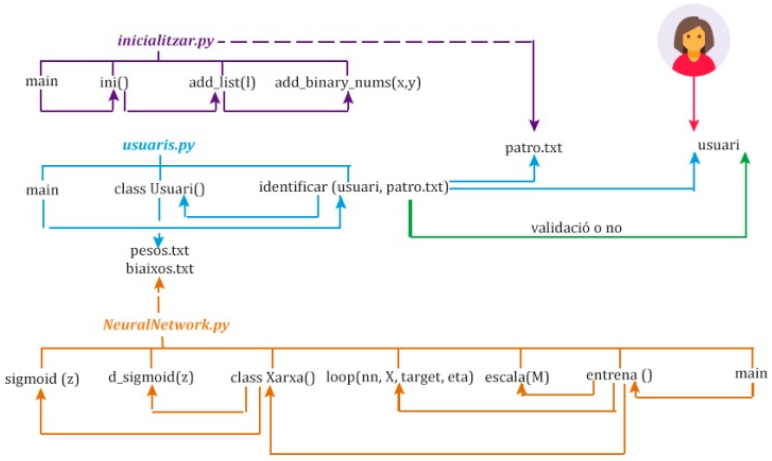


Figura 3.21. Jerarquia de les funcions (Font: Pròpia amb icones extretes del recurs https://icons8.com/icons)

En una primera instància, es crida al fitxer inicialitzar.py que rep el patró via USB i el guarda en un fitxer anomenat patro.txt.

Seguidament, es crea i s'entrena la xarxa neuronal que és única per cada persona a identificar, s'implementa el codi en NeuralNetwork.py i aquest fitxer crea fitxers per cada matriu de pesos i biaixos de la següent forma: una matriu per cada connexió.

Per últim, si es vol identificar un usuari, el fitxer usuaris.py demana un usuari a la persona que desitja identificar-se i busca els fitxers que guarden el patró, les matrius dels biaixos i les matrius dels pesos. Aquest fitxer també és el responsable de validar o no la contrasenya i comunicar-li a l'usuari.

Si s'entra en més detall en els fitxers encarregats de la creació, entrenament i validació de l'usuari, és a dir, NeuralNetwork.py i usuaris.py se'n pot extreure la següent informació:

El codi en NeuralNetwork.py crida a la implementació de la funció *entrena()*. Aquesta funció engloba tres funcions: *escala (M)*, *class Xarxa()* i *loop(nn, X, target, eta)*. La primera s'encarrega d'escalar les dades d'entrenament contingudes en una matriu M. La segona és la definició de la xarxa neuronal. El programa permet la creació de qualsevol xarxa neuronal, utilitzant *classes*. Els objectes, en la programació, estan formats per un conjunt de variables i funcions, aquests són creats mitjançant les classes. Per tal de crear una xarxa neuronal s'utilitza la classe Xarxa aquesta ha de rebre una llista amb tantes posicions com capes de la xarxa i en cada posició el número de neurones que aquesta conté. Per tant, en la xarxa neuronal de l'exemple de l'apartat anterior, la llista per poder crear-la és: [2, 3, 1], dues neurones d'entrada, tres neurones en la capa oculta i una neurona en la capa de sortida. Un objecte de la classe Xarxa, té quatre propietats: número de capes: *self.num_layers* (retorna un enter), número de neurones que té cada capa: *self.sizes* (retorna una llista), una llista amb les matrius que defineixen els pesos entre capes inicialitzats de forma aleatòria: *self.weights* (retorna una matriu) i una llista amb les matrius que defineixen els biaixos entre capes inicialitzats de forma aleatòria: *self.biases* (retorna una matriu). Cal notar que per a la definició aleatòria dels pesos i els biaixos es fa servei de la llibreria *Numpy* que ens permet obtenir nombres aleatoris dins d'una distribució normal.

A part de les propietats, un objecte d'aquesta classe presenta tres mètodes: *feedforward, backpropagation i valor_predit*. Aquests són, respectivament, l'algorisme de *feedforward*,

l'algorisme de *backpropagation* i una funció que retorna el valor predit per la xarxa davant d'un input.

En el primer mètode, s'implementa l'equació 33 de *feedforward*, en el segon les equacions de back propagation (39, 40, 41 i 42) i en l'últim mètode simplement es retorna la matriu d'activació de l'última capa, per tant, es retorna una part del mètode de *feedforward*.

La tercera funció utilitzada per *entrena()*és la funció *loop*, aquesta té per funció realitzar el bucle principal per un nombre concret d'iteracions, per tal que la xarxa realitzi l'aprenentatge i estigui preparada per tal de fer una predicció.

Amb una visió pràctica, cada usuari tindrà una xarxa neuronal personalitzada creada per la funció *entrena()* del fitxer NeuralNetwork.py i quan aquest es vulgui identificar, caldrà un identificador per tal que el programa busqui quina xarxa neuronal ha estat entrenada per identificar a l'usuari en qüestió mitjançant la funció *identificar(nom, patro)* continguda en usuaris.py.

Respecte al fitxer de lectura de signatura feta per l'usuari, es poden distingir tres funcions definides: *ini(), add_list(l) i add_binary_nums(x, y)*. Començant pel primer, es demana l'usuari i si l'usuari consta dins la base de dades s'envia el valor *108* al hardware, comencen a arribar dades, si la primera correspon a un 18 la signatura és vàlida i seguidament es poden llegir 201 valors més. Si altrament, el primer valor correspon a 81, la signatura no és vàlida i cal tornar a demanar-la enviant el valor 75, aquesta operació es podrà realitzar fins a 4 vegades seguides, si a la quarta la contrasenya no es correcta, es torna a demanar l'usuari.

A continuació s'adjunta una taula on es poden veure els codis de transmissió emprats entre hardware-software amb el seu significat.

Codi	Significat
El software envia un 108	Software preparat per la recepció de dades
18 en la primera dada rebuda pel software	Signatura vàlida
81 en la primera dada rebuda pel software	Signatura no vàlida
El software envia un 75	Petició de repetir contrasenya

Taula 3.1. Taula resum dels codis entre hardware i software.

3.6. Experimentació

A continuació es mostra la implementació de la solució en casos pràctics, així com altres implementacions de codis més senzills que ajuden a entendre la capacitat de les xarxes neuronals.

3.6.1. Experimentació amb un perceptró

En un inici, per entendre el funcionament intern d'una neurona artificial es va escriure un codi molt senzill per tal de donar solució al següent problema, a continuació es mostren els resultats obtinguts mitjançant aquest codi, però no s'entra en detall en l'explicació del seu funcionament, si és d'interès, es pot trobar el codi en l'Annex B.

X	Y
0 0 1	0
1 1 1	1
1 0 1	1
0 1 1	0

La neurona rep tres inputs (X) i ha de retornar la columna de sortida (Y). Si es para a reconèixer patrons entre l'entrada i la sortida, es pot apreciar que la sortida coincideix amb la primera columna de les X.

Per veure la capacitat d'aprendre que té la xarxa, es prova a fer 5 iteracions amb la neurona, es fa un *feedforward* i s'obtenen uns resultats com els de Y':

X	Y'
0 0 1	0,24760113
1 1 1	0,79671309
1 0 1	0,79248804
0 1 1	0,25245535

Si repetim la prova per 100 iteracions s'obté Y'':

X	Y''
0 0 1	0,07544325
1 1 1	0,93878595
1 0 1	0,95020356
0 1 1	0,0615456

En aquest cas es pot apreciar que la neurona ja ha après ja que més o menys els valors predits es corresponen als objectiu (Y).

Si es prova d'introduir a la xarxa els següents valors per veure la seva predicció:

X	Y
0 0 0	0
1 0 0	1
0 1 0	0
1 1 0	1

Els valors predits pel perceptró són els següents:

X	Y''
0 0 0	0,5

1 0 0	0,99579143
0 1 0	0,44556757
1 1 0	0,99476853

S'observa que aquest model aproxima molt bé els casos en què la resposta ha de ser 1 però en els casos que ha de donar 0 aquesta sortida no és tan evident, per tant, caldria fer més iteracions en l'entrenament o bé ampliar l'estructura per tal que el resultat sigui més fiable.

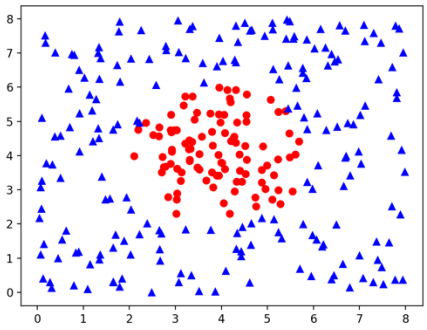
Tot i les observacions fetes per aquest model, es comprova la capacitat de resolució d'un sol perceptró: aquest és capaç de dur a terme una funció de tres variables amb un sol element, a més, cal remarcar el bon seguiment de patrons fet per aquest model ja que, en el cas anterior, la primera i la segona resposta desitjada són iguals i la segona amb la quarta també ho són. Deixant que el perceptró en faci la predicció es pot percebre de forma molt clara aquest patró.

Un cop vista la capacitat de resolució d'un sol perceptró, es pot passar al següent pas que és el codi de la xarxa neuronal que resol el problema plantejat.

3.6.2. Experimentació amb una xarxa neuronal per la resolució d'un problema de classificació

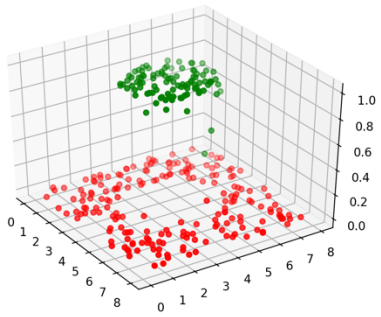
Un cop obtingut un programa que permetés la creació i entrenament d'una xarxa neuronal s'ha realitzat la pràctica següent per tal de verificar el funcionament correcte del codi.

Es defineixen un conjunt de punts aleatoris sobre d'un cercle en dues dimensions i se'n distingeixen dos tipus: els valors que es troben dins del cercle prenen per valor 1 i els que són fora del cercle prenen per valor 0. Mitjançant una xarxa neuronal creada de la mateixa forma que la xarxa per la identificació de l'usuari, aquesta tindrà quatre capes, la primera d'entrada amb dos neurones artificials, corresponent a cada dimensió de l'espai en què es representa el cercle, 5 neurones en la primera capa oculta, 2 en la segona capa oculta i una en la capa de sortida. S'entrena la xarxa a partir de valors aleatoris categoritzant aquells punts que estiguin fora del cercle amb un valor desitjat d'u i la resta amb un valor desitjat de zero.



Gràfic 2.22. Punts aleatoris en un gràfic de dues dimensions. (Font: pròpia)

S'entrena la xarxa mitjançant l'algorisme definit en el codi i es fa un gràfic on els eixos x i y corresponen als punts aleatoris amb els quals s'entrena i en l'eix z es gràfica el valor desitjat, aquest mostra la següent informació:



Gràfic 3.23. Gràfic dels punts aleatoris amb el valor predit per la xarxa neuronal. (Font: pròpia)

En el gràfic s'observa el bon funcionament de la xarxa neuronal descrita ja que en la majoria de punts fa una bona predicció, és cert que algun dels punts falla però no és la tendència.

3.6.3. Experimentació amb una xarxa neuronal per a la identificació d'un usuari

Dades d'entrenament

En primer lloc, s'entra en detall amb les dades utilitzades per dur a terme aquesta experimentació. Es tracta de 35 contrasenyes corresponents a l'usuari a identificar. Es disposa

de 40 contrasenyes entre les quals en podem distingir 20 corresponents a un altre usuari i 20 d'aleatòries.

En total es tracta de 75 dades d'entrenament, aquestes dades han estat realitzades de forma que s'adaptin a la realitat, és a dir: de forma ràpida, lenta, fent menys pressió sobre els sensors, aixecant dits durant la seqüència, entre d'altres. Aquestes dades ajuden a tenir un model més realista i entrenant la xarxa amb dades d'aquest tipus permet obtenir un sistema d'identificació molt dinàmic.

Les 75 contrasenyes es tenen guardades en format .txt, cada fila del text correspon a una contrasenya, les 35 primeres corresponen a l'usuari a identificar i les 40 següents a altres. Es pot veure el text que conté les dades que entrena l'algorisme en l'Annex C.

Per cada contrasenya, el codi està programat per fer la transformació del text en una matriu on cada fila correspon a una contrasenya i cada columna conté 100 valors, corresponents a els valors enviats pel hardware i l'assignació del vector de sortida desitjat amb 35 valors de sortida amb valor 1, corresponents a l'usuari a identificar i 40 valors 0 corresponents a l'altre usuari.

Dades de validació

Pel que fa a les dades d'identificació, rebudes a directament del sensor, es disposa de 5 contrasenyes. Aquestes a la pràctica estaran formades per 100 valors i un valor més addicional que realitzarà la suma binaria directa dels 200 valors enviats per tal de validar que la transferència de dades ha estat correcta. Cal recordar que tot i tractar-se de contrasenyes formades per 100 valors de 16 bits, aquestes arriben en dues parts de 8 bits cadascuna.

Per tal de facilitar l'experimentació, com que en aquest cas es vol parar compte amb el processament de les dades i la fidelitat de la xarxa, aquestes dades seran guardades amb fitxers sense el valor de la suma.

Estructura de la xarxa neuronal

Estructura 1: 29 neurones en la primera capa oculta i 6 en la segona

Es recorda que l'estructura de la xarxa neuronal, s'ha decidit anteriorment que sigui: 100 valors en la capa d'entrada, 29 neurones en la primera capa oculta, 6 neurones el la segona capa oculta i una neurona en la capa de sortida. A continuació s'han realitzat proves amb valors de neurones en les capes ocultes similars per tal de veure la fidelitat de cadascuna de les estructures d'aquesta manera, s'elegeix aquella que s'adapti més a les dades.

S'implementa el codi d'acord amb la descripció feta en els apartats anteriors. En una primera part, s'entrena la xarxa i es retorna el valor del vector predit per la xarxa neuronal, aquest hauria de ser similar a un vector amb 35 uns en la primera part del vector i 40 zeros en la segona part. El període d'entrenament es farà en 1.000.000 d'iteracions.

Ara es validen les 5 contrasenyes i s'obtenen els següents resultats:

Contrasenya	**Valor predit**	**Error relatiu [%]**
1.1	0,99745881	0,25
1.2	0,99645546	0,35
1.3	0,99816676	0,18
1.4	0,9979571	0,2
1.5	0,99738779	0,26

Taula 3.1. Valors predits i error relatiu per una xarxa neuronal amb 35 neurones en dues capes ocultes

Per tal de poder verificar si aquesta xarxa és l'òptima es repeteix el procediment amb les següents combinacions de neurones ocultes ja que aquest s'ha arrodonit de les expressions matemàtiques (3) i (4).

Estructura 2: 28 neurones en la primera capa oculta i 6 en la segona

Contrasenya	**Valor predit**	**Error relatiu [%]**

1.1	0,99801437	0,19
1.2	0,99084345	0,91
1.3	0,99936849	0,063
1.4	0,99919537	0,08
1.5	0,99939409	0,06

Taula 3.2. Valors predits i error relatiu per una xarxa neuronal amb 34 neurones en dues capes ocultes

Estructura 3: 28 neurones en la primera capa oculta i 5 en la segona

Contrasenya	**Valor predit**	**Error relatiu [%]**
1.1	0,99814733	0,18
1.2	0,83712877	16,29
1.3	0,99843197	0,16
1.4	0,99833747	0,17
1.5	0,99826249	0,17

Taula 3.3. Valors predits i error relatiu per una xarxa neuronal amb 33 neurones en dues capes ocultes

Estructura 4: 27 neurones en la primera capa oculta i 6 en la segona

Contrasenya	**Valor predit**	**Error relatiu [%]**
1.1	0,9987355	0,12
1.2	0,26262657	73,74
1.3	0,99914635	0,085
1.4	0,99935756	0,064
1.5	0,99700072	0,3

Taula 3.4. Valors predits i error relatiu per una xarxa neuronal amb 33 neurones en dues capes ocultes

A partir dels resultats obtinguts en les quatre estructures, es pot apreciar que aquelles que són capaces de realitzar prediccions més acurades són l'estructura 1 i la 2. Per la resta d'estructures alguns dels valors divergeixen de forma molt clara. Si s'observen ambdós resultats obtinguts mitjançant les estructures 1 i 2, es pot veure que la segona estructura és capaç d'acceptar contrasenyes amb un error relatiu menor que el que es capaç l'estructura 2.

Finalment, entre les estructures 1 i 2, s'elegeix aquella que ofereix un error relatiu menor, per tant, la que s'utilitzarà com a software final és l'estructura 2, la qual correspon a 28 neurones en la primera capa oculta i 6 en la segona.

Observant els errors relatius resultants de la xarxa elegida, es pot apreciar que acceptant un 5% d'error relatiu com a vàlid, la xarxa dona bons resultats. Aquest error podria ser més petit però acceptem petites variacions de les prediccions resultants de patrons menys clars o amb informació incomplerta.

Nombre d'iteracions

Per tal d'elegir l'estructura s'ha utilitzat un període d'entrenament format per 1.000.000 iteracions, a continuació es busca el temps òptim garantint que els valors validats en l'apartat anterior no sobrepassin el 5% d'error relatiu.

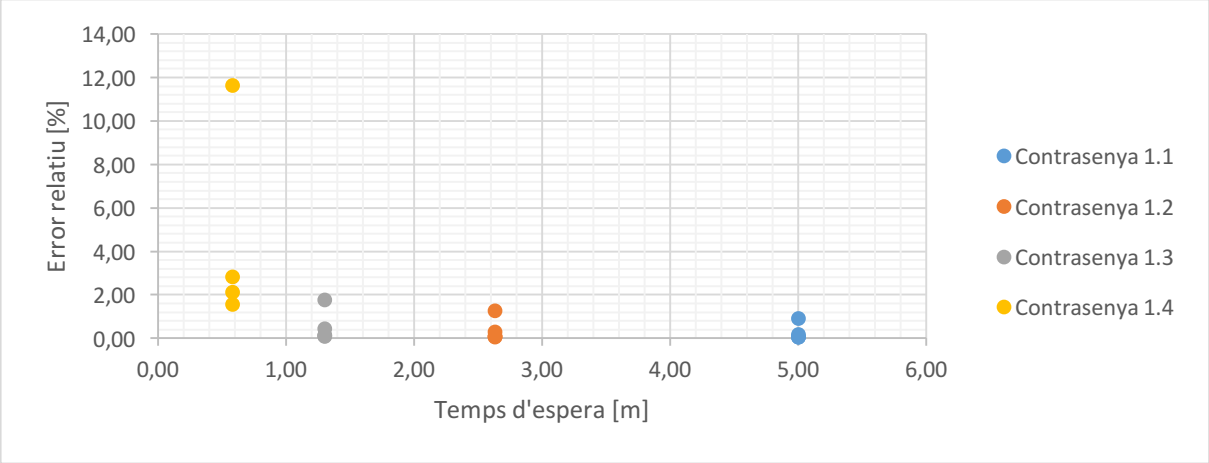
Per fer-ho es calcula el temps que tarda el programa en entrenar la xarxa neuronal, cal tenir en compte que l'estructura del programa ha estat realitzada amb cura i intentant que el programa sigui el més eficient possible (per exemple, sempre que ha estat possible s'ha utilitzat bucles realitzats amb *for* i no amb *whiles*) i sempre que s'ha pogut s'han utilitzat funcions predefinides (per exemple *len()* o *zip()*). Per últim, cal afegir que les proves s'han realitzat amb un Macbook Pro amb el següent processador: 2,6 GHz Intel Core i5.

Experiment	**Número iteracions**	**Temps d'espera [min]**	**Error relatiu [%]**
1	1.000.000	5	0,19
			0,91
			0,063
			0,08
			0,06

2	505.000	2,63	0,29
			1,27
			0,089
			0,11
			0,08
3	257.500	1,3	0,44
			1,77
			0,13
			0,16
			0,11
4	10.000	0,58	7,3
			11,64
			2,13
			2,84
			1,57

Taula 3.5. Experiment amb el nombre d'iteracions.

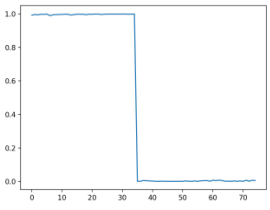
Amb les dades obtingudes, s'arriba a la conclusió que l'experiment número 4 ofereix un error que sobrepassa el límit marcat anteriorment i es descarta, a continuació es fa el següent gràfic per tal d'ajudar a l'elecció de de les iteracions:



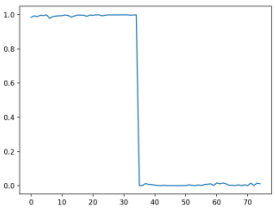
Gràfic 3.1. Error relatiu en funció del temps d'espera

Veiem com el temps disminueix de forma dràstica mentre els canvis que es produeixen en els errors relatius són molt petits. Com ja s'ha dit, es descarta l'experiment número 4 i veient el gràfic s'arriba a la conclusió que cinc minuts d'entrenament està sobredimensionat en relació als erros relatius que s'obtenen amb moltes menys iteracions. Finalment, es decideix que s'utilitzaran 257.500 iteracions ja que 1,3 minuts és un temps raonable i els errors que produeix entrenar les dades durant aquest període no afecta a la decisió de validar o no una contrasenya.

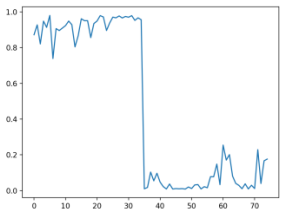
Un altre aspecte interessant a comprovar és l'aproximació de les dades entrenades depenent del número d'iteracions.



Gràfic 3.2. Punts d'entrenament en 1.000.000 iteracions



Gràfic 3.3. Punts d'entrenament en 275.500 iteracions



Gràfic 3.4. Punts d'entrenament en 10.000 iteracions

Mentre amb un nombre molt elevat d'iteracions s'obtenen gairebé línies rectes de punts d'entrenament, amb un nombre molt petit d'iteracions els punts d'entrenament difereixen molt entre ells. Cal remarcar que el més important és que hi hagi una clara distinció entre els punts d'entrenament que validen a l'usuari i aquells que no ho fan, en els tres casos es pot apreciar aquesta afirmació, evidentment, no es pot acceptar un cas en el que hi ha tantes fluctuacions per una petita diferència de temps.

Efectivament, el cas elegit és correcte ja que aquest és obtingut en un període raonable de temps sense veure compromesa l'aproximació de les dades d'entrenament feta per la xarxa neuronal.

Coeficient d'aprenentatge

Una altra forma d'optimitzar la rapidesa i precisió és recalcular el coeficient d'aprenentatge. Durant les experimentacions anteriors s'ha utilitzat un valor constant de $\eta = 0{,}001$, es recorda que es modifica aquest valor en el cas que la xarxa neuronal no convergeixi o bé el temps d'espera sigui molt elevat. Com no es donen cap dels dos casos, es segueix utilitzant el mateix valor de coeficient d'aprenentatge constant.

Si es donés el cas d'entrenar un grup de dades d'entrenament molt elevat caldria fer modificacions per tal d'optimitzar la funció, per exemple, adaptar el coeficient segons si s'aproxima al mínim.

4. Resultats

A partir de les dades obtingudes en l'apartat anterior, s'obtenen els següents resultats.

En primer lloc, l'estructura de cada xarxa neuronal consta de quatre capes, la capa d'entrada està formada per 100 neurones artificials, la primera capa oculta per 28, la segona capa oculta de 6 i, finalment, la capa de sortida amb una neurona artificial.

En segon lloc, s'entrena la xarxa neuronal mitjançant 35 dades obtingudes d'un patró de pressió concret, representat en la taula 4.1.

Dit A	Dit B	Dit C	Dit D	Dit E
	X	X		
		X	X	
			X	X
		X	X	
	X	X		
X				
	X			
		X		
			X	
				X

Taula 4.1. Patró de pressions representat en una taula. (Font: pròpia)

Juntament amb un segon conjunt de 40 dades aleatòries que s'han fet sense pensar en cap patró concret.

Després de fer l'entrenament, el valor del vector de sortides desitjades amb el que la xarxa neuronal entrena és el següent:

Y'=

[[9.83795727e-01 9.92434910e-01 9.88149684e-01 9.96283923e-01
9.94229309e-01 9.98616001e-01 9.78193125e-01 9.88231655e-01
9.91199844e-01 9.92787398e-01 9.93723883e-01 9.97395334e-01
9.95014366e-01 9.85165800e-01 9.91752423e-01 9.97177479e-01
9.96663209e-01 9.96259514e-01 9.90186447e-01 9.96956814e-01
9.95683746e-01 9.98682670e-01 9.98209136e-01 9.92694050e-01
9.96721836e-01 9.98114600e-01 9.98035821e-01 9.98365451e-01
9.97873009e-01 9.98467216e-01 9.98315155e-01 9.98703790e-01
9.96935869e-01 9.97632039e-01 9.98058894e-01 9.80791384e-04
1.72915555e-03 1.31770773e-02 8.13085490e-03 6.70430226e-03
3.86046403e-03 1.65534416e-03 9.21314422e-04 3.10513810e-03
9.09319239e-04 9.73081225e-04 9.62915100e-04 1.02859147e-03
9.09400049e-04 1.35016414e-03 9.83773026e-04 5.18202991e-03
2.68528780e-03 9.75420677e-04 4.41186517e-03 1.38381733e-03
7.27831521e-03 8.84784853e-03 1.14780847e-02 2.34227793e-03
1.65601335e-02 1.12558019e-02 1.62508325e-02 1.14190780e-02
2.68640519e-03 2.77359441e-03 9.23387719e-04 5.33657434e-03
9.96625517e-04 5.52748248e-03 1.02115208e-03 1.51520314e-02
2.05551592e-03 1.47687378e-02 1.17853976e-02]]

Efectivament, es poden distingir de nou 35 primers valors pròxims al valor desitjat 1 i 40 següents amb valors pròxims a 0.

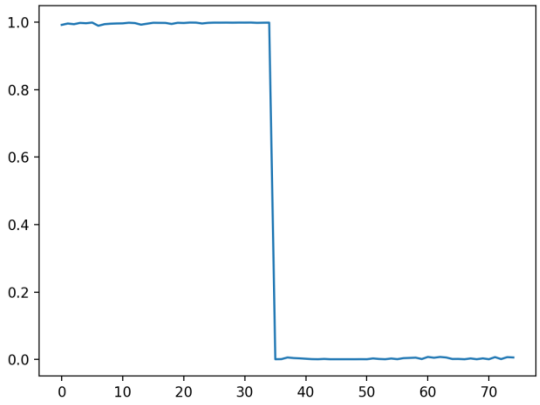


Figura 4.1. Valors de les dades d'entrenament predites per la xarxa. (Font: pròpia)

En aquest es pot observar de forma més clara com els valors que són pròxims a 1 gairebé no fluctuen, al igual que els valors propers a 0 i estan clarament diferenciats uns d'uns altres. S'observen alguns pics en els valors, aquests són deguts a punts d'entrenament que, tot i mostrar patrons semblants a la resta, difereixen més que la resta.

En ser dades experimentals és normal que es doni lloc a aquest tipus de fluctuacions, el més important es que la xarxa neuronal en distingeixi els dos grups per tal de poder fer la validació posterior.

Un cop la xarxa està entrenada es fa servei del codi per tal de veure com es comporta la xarxa davant de situacions que es poden donar en la lectura del patró a partir dels sensors, tots aquests casos es recullen en la taula 4.1. :

Casos	Valor predit	Usuari identificat?	Resultat esperat?
L'usuari reprodueix la contrasenya sense el dit central	0,15779162	No	No
L'usuari reprodueix bé la contrasenya però no fa les pulsacions del dit petit	0,91213721	No	No
L'usuari reprodueix la meitat de la contrasenya lentament i l'altra meitat ràpidament	0,00174582	No	No
L'usuari reprodueix tota la contrasenya de forma molt suau	0,00359338	No	Sí
L'usuari reprodueix la meitat de la seqüència igual i la segona meitat és inventada	0,68015134	No	Sí
L'usuari reprodueix la meitat de la seqüència igual i la segona meitat és inventada	0,00093103	No	Sí
L'usuari reprodueix el patró correcte dues vegades seguides	0,016773	No	Sí
L'usuari reprodueix la meitat de la contrasenya i deixa els dits sobre els	0,97583399	Sí	Sí

sensors sense aplicar pressió			
L'usuari reprodueix bé el patró però no fa les pulsacions el dit gros	0,99891993	Sí	Sí
L'usuari reprodueix la contrasenya correcta però molt ràpida	0,96766943	Sí	Sí
L'usuari representa la meitat del patró i aixeca els dits	0,98185518	Sí	No

Taula 4.2. Casos de contrasenyes que validen o no l'usuari.

Com es pot observar, la xarxa neuronal, hi ha casos en que està preparada per validar canvis en la contrasenya, per exemple, si el dit gros no marca pulsacions, si la contrasenya s'introdueix de forma ràpida, si es reprodueix la meitat de la contrasenya, però en altres ocasions no ho fa, per exemple, si manca algun dit per fer pulsacions, si es reprodueix la contrasenya a diferents velocitats, entre d'altres.

Alguns d'aquests resultats són esperats ja que la base de dades que s'ha utilitzat és limitada i no s'han dut a terme combinacions que entrenant-les seria més fàcil que la xarxa en reconegués el patró. Ja que si la xarxa ha "aprés" que l'usuari no ha fet cap entrenament polsant fluix sobre els sensors, prediu que no podrà ser ell encara que el patró pugui ser similar.

5. Planificació temporal i costos

Per tal de realitzar la planificació temporal, s'ha de tenir en compte els coneixements previs per conèixer el punt de partida. Com a formació prèvia es requereix un coneixement mitjà de programació amb llenguatge Python corresponent a un any d'aprenentatge de programació a

l'Escola Tècnica Superior d'Enginyers Industrials de Barcelona més aprenentatge autònom sobre les llibreries utilitzades com Numpy i Time i d'altres recursos per realitzar la lectura de dades a partir d'un port.

Partint del coneixement previ especificat, es realitza la planificació temporal del projecte en les següents etapes:

1. Recerca i estudi de l'estat de l'art end'Aprenentatge automàtic (5 setmanes).
2. Programació del software encarregat de la creació de xarxes neuronals (8 setmanes).
3. Programació del software responsable de la identificació de l'usuari (3 setmanes).
4. Programació del software de lectura de dades (1 setmanes).
5. Optimització del codi (6 setmanes)
6. Redacció de la memòria (14 setmanes)
7. Revisió (1/2 setmana)

Cada pas, té la durada especificada en parèntesis. Amb aquesta informació es realitza un diagrama de Gantt per ajudar a la planificació des del mes de setembre fins al mes de febrer.

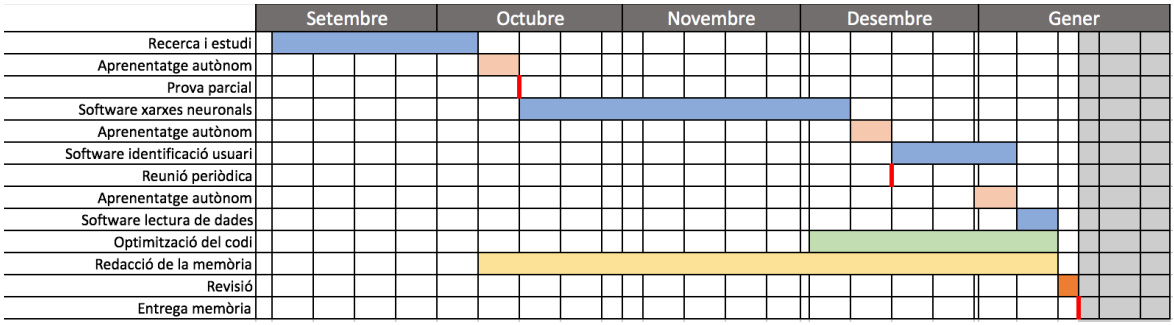


Figura 5.1. Diagrama de Gantt

En aquest, es poden observar les diferents etapes esmentades anteriorment així com els períodes d'aprenentatge autònom i els esdeveniments puntuals com són: prova parcial, reunió periòdica i entrega de la memòria.

Si es fa el recompte s'obtenen un total de 19,5 setmanes invertides en la realització del treball de fi de grau, si es té en compte el treball de 12 hores per cada setmana, s'obté un total de 234 hores de treball.

Amb la informació anterior es fa una estimació dels costos que suposa la realització d'aquest treball, el pressupost és el següent:

Quantitat	Temps emprat [mesos]	Temps d'amortització [mesos]	Unitats	Descripció	Preu unitari	Total
Costos fixos del desenvolupament						
1	5	60	0,083	**Ordinador** Macbook Pro amb processador: 2,6 GHz Intel Core i5	1500 €/u	125,00 €
Costos derivats del personal						
Quantitat	Unitats de mesura					
234	hores			**Estudiant d'enginyeria industrial** Hores dedicades al desenvolupament del projecte	10 €/hora	2.340,00 €
50	hores			**Enginyer industrial** Revisió del projecte	50 €/hora	2.500,00 €
Costos de serveis						
				Altres Despesa relacionada amb el consum de llum, d'aigua, ús d'altres dispositius com USB	500 €	500,00 €

TOTAL	**5.465,00 €**

S'ha suposat un estudiant d'enginyeria industrial per a la realització del projecte, realitzant les hores estimades pel desenvolupament del software i un enginyer industrial encarregat de la revisió del treball i la signatura del projecte.

En total, el cost del projecte és de 5.465,00 € en total.

6. Impacte medi ambiental

Pel que fa a l'impacte mediambiental, al tractar-se d'un software aquest és reduït. Cal pensar però amb el cost de la producció d'electricitat energia emprada pels dispositius electrònics que s'han utilitzat per a la realització del treball.

S'investiga l'origen de l'energia elèctrica a Catalunya, s'utilitzen dades del 2017 ja que no s'han trobat de més actuals però es considera que la variació entre 2017 i 2018 no serà gaire considerable.

Les dades de l'institut català d'energia afirmen que en l'any 2017 la font d'energia més utilitzada és la nuclear amb un 53,5 %. Aquest percentatge, sumat al 17,4% d'energia elèctrica obtinguda mitjançant els cicles combinats i el 11, 7% produïda per la cogeneració, dona un resultat de l'ús d'un 83,6% d'energies no renovables [7].

Aquests percentatges mostren que l'ús d'aparells electrònics i, per tant, d'electricitat suposen un cost al medi ambient. Per tant, cal que cada usuari consumidor d'electricitat en sigui conscient i eviti males pràctiques per tal de reduir l'ús d'electricitat.

Exemples de bones pràctiques en aquest projecte poden ser: apagar la pantalla de l'ordenador quan no es faci servir, ja que és la part que més consumeix d'un ordenador, apagar els dispositius electrònics quan no s'utilitzin per tal d'estalviar bateria, utilitzar l'ús de baix consum dels dispositius electrònics sempre que aquests en disposin, entre d'altres.

Conclusions

Finalment, tenint en compte els objectius plantejats a l'inici d'aquest projecte es conclou:

- S'ha desenvolupat un software capaç de reconèixer identitats. Aquest software ha estat programat amb llenguatge Python amb l'ajuda de llibreries del mateix llenguatge. El software executa l'algorisme d'aprenentatge supervisat basat en les xarxes neuronals. Aquest "aprèn" de l'experiència mitjançant la minimització de l'error en les prediccions incloent canvis en els seus paràmetres calculats a partir del descens pel gradient.

- El sistema és capaç de reconèixer un patró per cada usuari i el pot diferenciar entre altres contrasenyes, tot i que pot cometre errors degut a la base de dades limitada amb què està entrenat l'algorisme.

- El sistema posseeix una adaptabilitat considerable ja que es capaç de reconèixer el mateix patró reproduït de diferents formes, tal i com s'ha vist pot distingir patró si manca la informació d'algun dit o bé si la seqüència s'ha realitzat de forma més ràpida.

- S'ha fet una introducció al xifrat de dades a través del mètode (RSA) tot i que no s'ha pogut implementar en el treball degut a la manca de temps. A la pràctica, però, s'ha de ser conscient de la importància de xifrar les dades per tal de garantir la seguretat de les dades de l'usuari.

- Per últim, s'ha realitzat un estudi profund dels algorismes d'aprenentatge supervisat, concretament, de les xarxes neuronals, així com dels detalls de la seva implementació emprant un llenguatge d'alt nivell. Entendre el funcionament intern de la xarxa ajuda a entendre els resultats que aquesta prediu i ajuda a la identificació de possibles errors.

Treball futur

Optimització de la implementació software

Com s'ha comentat en repetides vegades durant el projecte, s'han anat implementant optimitzacions per tal que el programa sigui el més eficient possible: emprant funcions predefinides, llibreries de Python, etc. L'augment de l'eficiència d'un software és complicada i sempre hi ha millores a realitzar per tal que el codi treballi el més ràpid possible.

Una altre aspecte que es pot modificar per tal d'assolir aquest objectiu és la implementació del coeficient d'aprenentatge per trams. A la pràctica es poden tenir milions de dades d'entrenament i per tant, tenir un codi d'aprenentatge fix frena el processament i allarga el temps d'espera per l'obtenció del resultat.

Ús d'una interfície gràfica

Per finalitzar, es pot plantejar l'ús d'una pantalla per tal de fer la interacció entre el software i l'usuari. En aquesta interfície l'usuari podrà veure què l'estat del codi. Per exemple, suposant que l'usuari s'està registrant i, per tant, cal entrenar una xarxa neuronal nova i aquest procés és lent, doncs mitjançant la pantalla l'usuari pot veure en quin estat es troba l'entrenament i el temps aproximat que queda per acabar l'operació. Un altre exemple seria la identificació de la persona, en la pantalla hi pot aparèixer les peticions a l'usuari: 'Introdueixi la contrasenya', 'Introdueixi l'usuari', 'Contrasenya correcta', 'Contrasenya incorrecta', entre d'altres.

Agraïments

En primer lloc, voldria agrair al director d'aquest treball, l'Álvaro Gómez, per l'ajuda i el seguiment proporcionats al llarg de tot el projecte. També al Salvador Manich, el codirector, per aportar un altre punt de vista sobre aquest projecte.

En segon lloc, a la meva família, pel suport durant el grau i durant la realització d'aquest projecte.

Per últim, agrair a les meves companyes i companys que m'han acompanyat durant tot el trajecte.

Referències

[1] J. Simske, S. (2010). *Dynamic biometrics: The case for a real-time solution to the problem of access control, privacy and security*. [online] Ieeexplore-ieee-org.recursos.biblioteca.upc.edu. Disponible a: https://ieeexplore-ieee-org.recursos.biblioteca.upc.edu/document/5507535 [Accés el 11 Des. 2018].

[2] Bishop, C. (2006). *Pattern Recognition and Machine Learning*.

[3] Y. LeCun and C. Cortes, "MNIST handwritten digit database", 2010. [Online]. Disponible a: urlhttp://yann.lecun.com/exdb/mnist/

[4] Hastie, T., Friedman, J. and Tibshirani, R. (2017). *The elements of statistical learning*. New York: Springer.

[5] Sato, K. (2019). *Using machine learning for insurance pricing optimization | Google Cloud Blog*. [online] Google Cloud Blog. Disponible a: https://cloud.google.com/blog/products/gcp/using-machine-learning-for-insurance-pricing-optimization [Accés el 17 Des. 2018].

[6] Huang, G. (2019). *Learning capability and storage capacity of two-hidden-layer feedforward networks - IEEE Journals & Magazine*. [online] Ieeexplore.ieee.org. Disponible a: https://ieeexplore.ieee.org/document/1189626 [Accés el 10 Gen. 2019].

[7] Institut Català d'Energia. (2017). *Balanç d'energia elèctrica de Catalunya*. [online] Disponible a: http://icaen.gencat.cat/ca/energia/estadistiques/resultats/anuals/balanc_energia/ [Accés el 12 Gen. 2019].

Annexes

A Codi

A.1. Codi NeuralNetwork.py

```
import numpy as np
import matplotlib.pyplot as plot
import usuaris

def sigmoid(z):
        return 1.0/(1.0+np.exp(-z))

def d_sigmoid(z):
        return np.exp(-z)/(1+np.exp(-z))**2

class Xarxa():

        def __init__(self, sizes):


                self.num_layers = len (sizes)
                self.sizes = sizes


                self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
                self.weights = [np.random.randn (y, x) for x, y in zip(sizes[:-1], sizes[1:])]

        def feedforward (self, X):
                A = []
                Z = []

                for b, w in zip(self.biases, self.weights):
                        z = np.dot(w,X)+b
                        a = sigmoid(z)
                        Z.append(z)
                        A.append(a)
                        X=a

                return A,Z


        def backpropagation (self, X, target, ff, eta):

                a_L = ff[0][-1]
```

```
z_L=ff[1][-1]

A=ff[0]

A.insert(0, X)

delta_L = (a_L - target) * d_sigmoid(z_L)


i = -2
delta_ant = delta_L
delta=[]
delta.append(delta_ant)


while abs(i)!= self.num_layers:

        z = ff[1][i]
        delta_l = (np.dot((self.weights[i+1]).T, delta_ant))*d_sigmoid(z)
        delta.append(delta_l)

        delta_ant = delta_l

        i -= 1

delta.reverse()


grad_W = []
grad_B = []

l=0

for d in delta:

        g_w = np.dot(d, A[l].T)
        grad_W.append(g_w)

        l += 1


for d in delta:

        r = np.sum(d, axis=1)
        g_b=r.reshape((len(r), 1))

        grad_B.append(g_b)
```

```
                self.weights = [w - eta * nw for w, nw in zip(self.weights, grad_W)]

                self.biases = [b - eta * nb for b, nb in zip(self.biases, grad_B)]

                return (self.weights, self.biases)

        def valor_predit(self, X):

                print('Aquest es el valor predit per la xarxa :')
                #plot.plot(self.feedforward(X)[0][-1][0])
                #plot.show()

                return self.feedforward(X)[0][-1]


def loop(nn, X, target, eta):

        for i in range(257500):

                ff = nn.feedforward(X)

                bp = nn.backpropagation(X, target, ff, eta)

        return nn.valor_predit(X)


entrenament=open('dades.txt', 'r')

password=[]
for contrasenya in entrenament
        contra = contrasenya.split('\t')
        l=[]

        for c in contra:
                l.append(int(c, 2))
        password.append(l)
X=np.array(password).T

Y=np.array([[1 for y in range(35)]+[0 for f in range(40)]])


def escala(M):
        minim=0
        maxim=65535
        X=[]
        for fila in M:
                x=[]
```

```
            for ele in fila:
                y=(ele-minim)/(maxim-minim)
                x.append(y)
            X.append(x)
        return [np.array(X), minim, maxim]



def entrena():

        entrenament=open('dades.txt', 'r')

        password=[]
        for contrasenya in entrenament:
            contra = contrasenya.split('\t')
            l=[]

            for c in contra:
                l.append(int(c, 2))
            password.append(l)
        X=np.array(password).T

        [inp, minim, maxim]=escala(X)

        Y=np.array([[1 for y in range(35)]+[0 for f in range(40)]])

        nn=Xarxa([100, 28, 6, 1])
        eta=0.001

        print(loop(nn, inp, Y, eta))

        return [nn, minim, maxim, nn.biases, nn.weights]


if __name__=='__main__':
        l=entrena()
        i=0
        j=0
        for ele in l[3]:
            if i<=len(l[3])-1:
                np.savetxt(f'usuari2.{i}_biases.txt', ele, fmt='%s')
                i+=1
            else:
                break

        for ele in l[4]:
```

```
if j<=len(l[4])-1:
        np.savetxt(f'usuari2.{j}_weights.txt', ele, fmt='%s')
        j+=1
else:
        break
```

A.2. Codi usuaris.py

```
import NeuralNetwork as NN
import time
import numpy as np

class Usuari():
        def __init__(self, nom):
                self.nom=nom
                self.xarxa=NN.Xarxa([100, 28, 6, 1])

                self.xarxa.biases = []
                self.xarxa.weights = []

                for i in range(3):
                        bias=np.loadtxt(f'{nom}.{i}_biases.txt')
                        bias=np.matrix(bias).T
                        self.xarxa.biases.append(bias)

                for j in range(3):
                        weight=np.loadtxt(f'{nom}.{j}_weights.txt')
                        self.xarxa.weights.append(weight)

def identificar (nom, patro):

        usu=Usuari(nom)

        minim=0
        maxim=65535

        contras=open(patro, 'r')

        contrasenya = contras.readline()


        contrasenya=contrasenya.split('\t')


        Contrasenya = []

        for ele in contrasenya:

                Contrasenya.append(int(ele, 2))

        x = np.array([Contrasenya]).T

        X=[]
        for fila in x:
```

```
            t=[]
            for ele in fila:
                y=(ele-minim)/(maxim-minim)
                t.append(y)
            X.append(t)
        X_ = np.array(X)


        f=usu.xarxa.valor_predit(X_)

        print(f)

        print ("Aquesta contrasenya és ")

        if f[0] >= 0.97:
            print ("correcta")

        else:
            print ("incorrecta")



if __name__=='__main__':
        nom=input("Hola!, introdueix l'usuari: ")

        identificar( nom, 'patro.txt')
```

A.3. Codi inicialitzar.py

```
import serial
import time


def ini():

        ser=serial.Serial('/dev/cu.SLAB_USBtoUART', 19200)

        ser.write(byte([108]))

        text=open('patro.txt', 'w')

        i=1

        k=0

        l=[]

        while True:

                tdada = ser.read()
                time.sleep(1)
                data_left = s.inWaiting()
                tdata += s.read(data_left)
                suma = tdata[-1]

                if tdata[0] == 81 and k<=4:
                        ser.write(byte([75]))
                        k+=1

                elif tdata[0] == 18:
                        text=open('patro.txt', 'w')
                        for ele in tdata:
                                l.append(bin(ele)[2:])
                                if i%2==0 and bin(ele)[2:][0] == '1':
                                        text.write(bin(ele[2:])+'\n')

                                        if suma != add_list(l) and k<=4:
                                                ser.write(byte[75])
                                                k+=1
                                        else:
                                                print('Contrasenya incorrecta')

                                elif i%2==0 and bin(ele)[2:][0] == '0':
                                        text.write(f+'\t'+bin(ele)[2:])+'\t')
                                        i+=1
```

```
                    elif i%2 != 0:
                        f=bin(ele)[2:]
                        i+=1

    ser.close()

    text.close()


def add_list(l):

    num3 = add_binary_nums (l[0], l[1])

    for num in l[2:]:

        num3 = add_binary_nums(num, num3)

    return num3


def add_binary_nums(x,y):   ##Suma directa
    max_len = 8

    result = ''
    carry = 0

    for i in range(max_len-1, -1, -1):
        r = carry
        r += 1 if x[i] == '1' else 0
        r += 1 if y[i] == '1' else 0
        result = ('1' if r % 2 == 1 else '0') + result
        carry = 0 if r < 2 else 1

    return result.zfill(max_len)
```

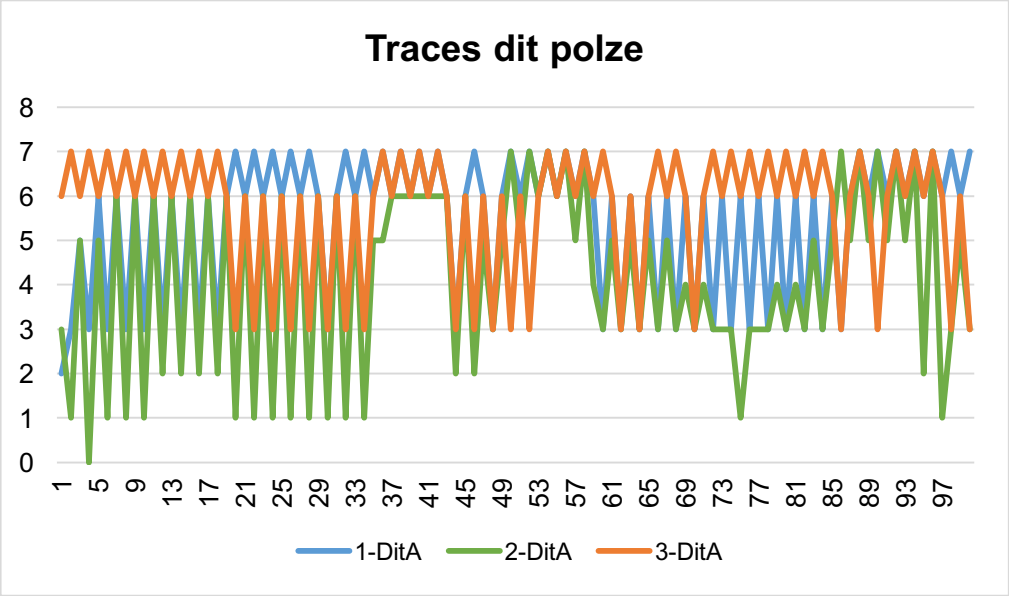
B Codi perceptron.py

```
import numpy as np

def sigmoid(x):
        return 1 /(1 + np.exp(-x))

def d_sigmoid(x):
        return x * (1 - x)

training_inputs = np.array([[0, 0, 1],
                            [1, 1, 1],
                            [1, 0, 1],
                            [0, 1, 1]])

training_outputs = np.array([[0, 1, 1, 0]]).T

np.random.seed(1)

weights = 2 * np.random.random((3, 1)) - 1

print ( 'Random starting weights: ')
print (weights)

for iteration in range(100):

        input_layer = training_inputs


        outputs = sigmoid(np.dot(input_layer, weights))


        error = (training_outputs - outputs) # z(L) - y

        adjustments = 2 * error * d_sigmoid (outputs)

        weights += np.dot(input_layer.T, adjustments)

print('Weights after training')
print(weights)


print('Outputs after training: ')
print(outputs)

inputs = np.array([[0, 0, 0],
                   [1, 0, 0],
```

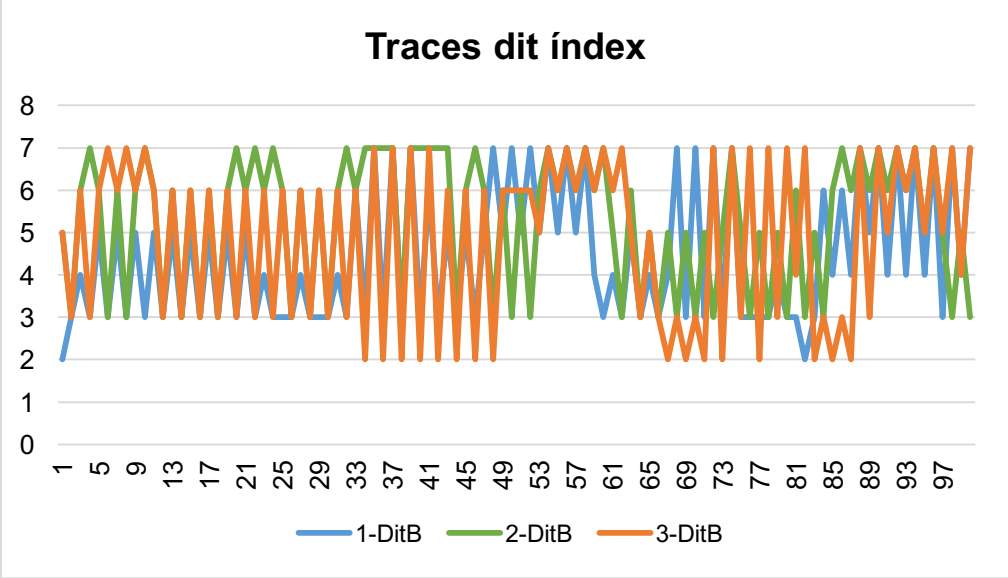
```
                    [0, 1, 0],
                    [1, 1, 0]])

prediccio = sigmoid(np.dot(inputs, weights))

print('Valor predit: ')
print(prediccio)
```

C Gràfic de traces

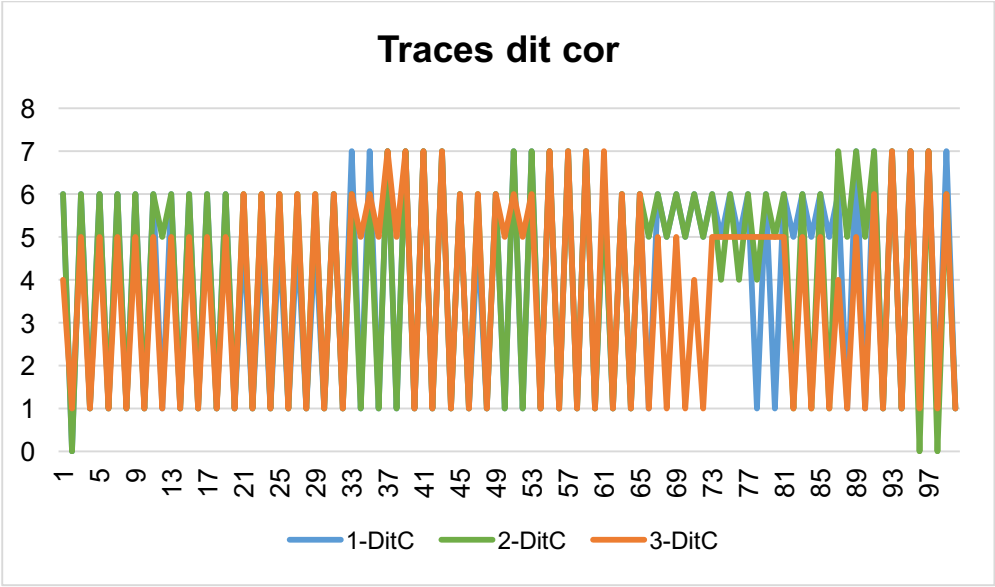
En aquest annex es mostra un conjunt de cinc gràfics que fan referència a cada dit de la mà, aquests gràfics pretenen donar la idea de les dades que tracta la xarxa neuronal. En l'eix Y està representat el valor en decimal que arriba del hardware, en general, els valors més alts corresponen a un valor de pressió més alt, mentre que els més baixos són els que s'han realitzat amb una pressió menor sobre els sensors. L'eix Y dóna la referència de quina dada s'està tractant, des de la primera dada que es rep fins la última, en total 100 dades per cada contrasenya. Cada gràfica mostra tres traces, corresponents a tres contrasenyes fetes per la mateixa persona.



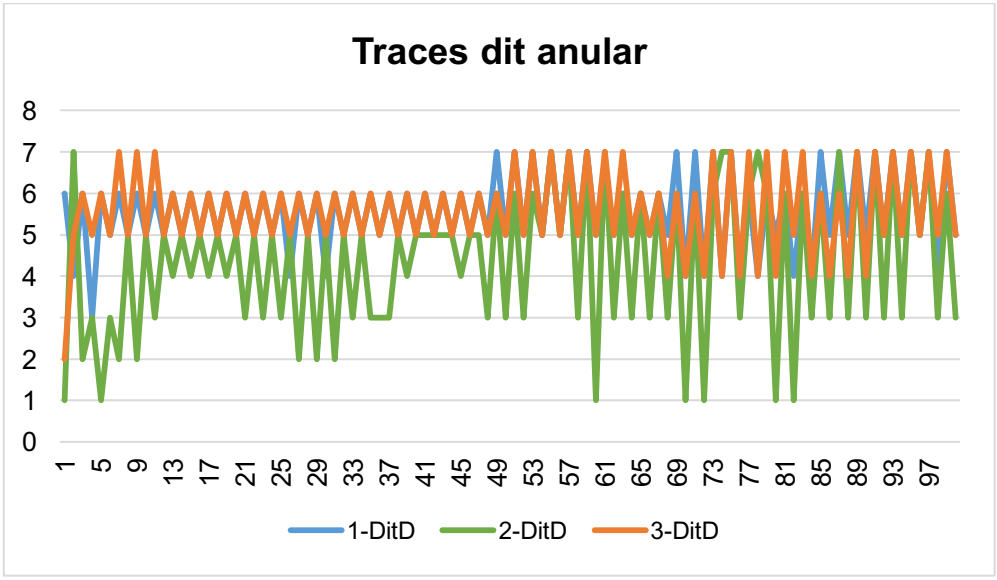
Gràfic B.1. Traces del dit polze. (Font: pròpia)



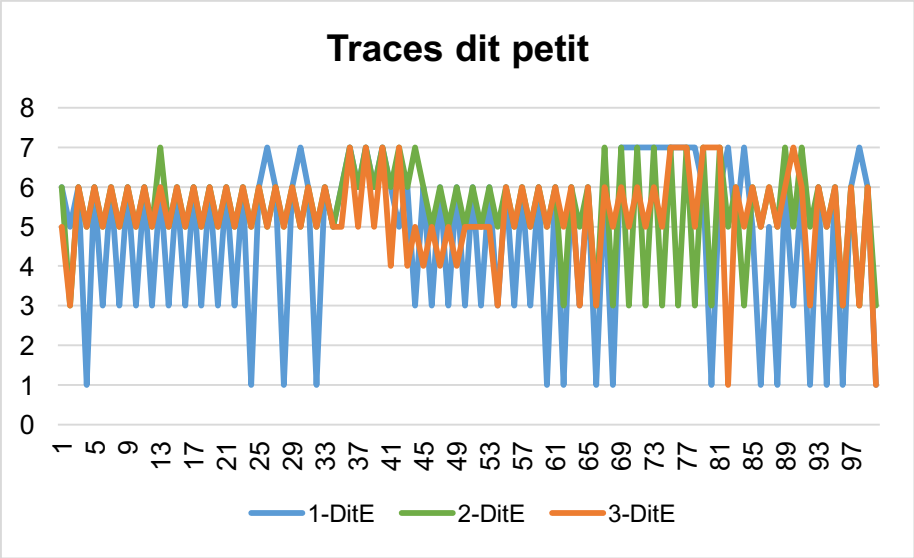
Gràfic B.2. Traces del dit índex. (Font: pròpia)



Gràfic B.3. Traces del dit cor. (Font: pròpia)



Gràfic B.4. Traces del dit anular. (Font: pròpia)



Gràfic B.4. Traces del dit petit. (Font: pròpia)